PROGRAMACIÓN DE LA PRODUCCIÓN EN UN SISTEMA FLEXIBLE FLOW SHOP UTILIZANDO LA METAHEURÍSTICA VIRAL SYSTEMS

Ing. Esp. JORGE MARIO LÓPEZ PEREIRA

Presentado como requisito para optar el título de Magister en Ingeniería Industrial

Director: M.Sc. CARLOS ARDILA HERNÁNDEZ.

UNIVERSIDAD DEL NORTE
DEPARTAMENTO DE INGENIERÍA INDUSTRIAL
MAESTRÍA EN INGENIERÍA INDUSTRIAL
BARRANQUILLA, COLOMBIA
2011

NOTAS DE ACEPTACIÓN

_____
_____
_____
_____
_____

_____
M.Sc. CARLOS ARDILA
Director del proyecto

_____
MSc. Elías Niño Ruiz

_____
Phd. Alcides Santander Mercado

AGRADECIMIENTOS

A Dios, fuente toda sabiduría, que siempre ilumina mi camino.

A mis padres, Lucy Pereira y Jorge Efraín Lopez por el apoyo incondicional que me dieron a lo largo de mi vida.

A mi Hija Adriana Sofía por permitirme soñar y crecer con su imaginación.

A Mi esposa Ingrid Cecilia Rosario, por acompañarme y asesorarme en todas los momentos de mi vida.

A mis hermanos Jorge Efraín y Alberto Alfonso, por su motivación constante y por convencerme que hiciera esta maestría.

A Jaime Hernandez, por enseñarme que no hay límites, que la única barrera somos nosotros mismos.

Al Ingeniero Carlos Ardila por su asesoría y dirección en el trabajo de investigación.

A mis compañeros docentes de la Universidad de Córdoba por guiarme y motivarme en el transcurso de la tesis.

Al ingeniero Helman Hernandez quien me mostró lo interesante que es la investigación.

Y a todas aquellas personas que colaboraron o me motivaron en la realización de esta investigación, hago extensivo mi más sincero agradecimiento.

TABLA DE CONTENIDO

Pag.

LISTA DE TABLAS

Pag.

LISTA DE ILUSTRACIONES

Pag.

LISTAS DE ECUACIONES

Resumen

En este trabajo, se implementa la metaheusitica Viral Systems [1] , con el fin de resolver el problema de programación de la producción en sistemas Flexible Flow Shop (FFS), con máquinas paralelas no relacionadas en cada estación y tiempos de preparación dependientes de la máquina y de la secuencia.

La finalidad es encontrar una adecuada secuencia de producción de las órdenes de trabajo, teniendo en cuenta los tiempos de liberación y procesamiento de las órdenes, los tiempos de preparación de las máquinas en cada estación y las fechas de entrega de cada trabajo, con el fin de minimizar el makespan (C_{max}).

Con el objetivo de examinar el desempeño de Viral system en este problema, se implementaron dos variaciones de esta metaheurística, se realizó un diseño experimental de superficie de respuesta para encontrar los mejores parámetros, luego, se comparó los resultados con el algoritmo genético propuesto por Ruiz y maroto 2006.

Los resultados arrojados muestran que la metaheurística Viral System con encoding permutado genera mejores resultados que los demás algoritmos evaluados y con tiempos de ejecución similares.

Palabras Claves: Programación de la producción, Flexible Flow Shop (FFS), Viral Systems, algoritmo genéticos.

Abstract

Keywords: Scheduling, Flexible Flow Shop, Viral Systems, Genetic Algorithm.

CAPÍTULO 1 : GENERALIDADES

1.1. INTRODUCCIÓN

En la presente investigación, se ha adaptado, aplicado y parametrizado una nueva metaheurística, denominada Viral Systems, la cual fue desarrollada como una herramienta para dar solución a problemas de optimización combinatoria con un solo objetivo, la cual se caracteriza por emular el comportamiento de sistemas víricos, además, permite identificar y mantener soluciones de alta calidad al problema que se intenta resolver.

En nuestro caso, el problema a resolver es el de programación de la producción en sistemas productivos Flexible Flowshop, y cuyo objetivo es la minimización del Lapso (Cmax), es decir la minimización del tiempo máximo de finalización de las órdenes.

En la solución del problema, se tienen en cuenta consideraciones como los tiempos de liberación, los tiempos de procesamiento de las órdenes en las diferentes máquinas que pertenecen a una misma estación, los tiempos de preparación de las máquinas en cada estación, los cuáles pueden ser dependientes de las máquinas o dependientes de la secuencia. Además de ello, no se admiten preemptions (interrupciones), lo cual indica que una vez iniciado el procesamiento de una determinada orden de producción en una máquina, ésta no puede ser detenida para dar paso a otra que es considerada de mayor importancia. Para efectos prácticos, se considera que la capacidad de almacenamiento entre máquinas consecutivas es ilimitada. Se pretende encontrar una secuencia adecuada de producción, que minimice el tiempo total de finalización.

La metaheuristica Viral Systems se han codificado y parametrizado en dos versiones. La primera de ellas, se utiliza un encoding random key, y la segunda, usa un encoding permutado y una heurística de inserción para la primera generación. Para los dos algoritmos Viral systems se realizó un diseño experimental de superficie de respuesta con el fin de encontrar los parámetros óptimos con los cuales debería ejecutarse. Estas versiones de Viral systems se comparan con un algoritmo genético propuesto en la literatura del cual se conoce sus buenos resultados. Las tres metaheuristicas se comparan en un diseño experimental, con el objetivo de determinar cuál de todos proporcionaba los mejores resultados. Todos los algoritmos fueron programados en el software MATLAB 2009b.

1.2. PLANTEAMIENTO DEL PROBLEMA

1.2.1. ANTECEDENTES

En la actualidad, y desde hace algunos años, las organizaciones se han venido enfrentando al fenómeno de la globalización, el cual las ha obligado de manera categórica a desarrollar nuevas formas de producción que les permita competir en un nuevo mercado, el cual se encuentra conformado por grandes organizaciones alrededor de todo el mundo, con una gran diversidad e innovación tecnológica que inexorablemente dejan rezagados a aquellos que no diseñen e implementen nuevas estrategias que les permitan hacer frente a tan voraz competencia.

Es de este modo, como las organizaciones alrededor de todo el mundo se han dado a la tarea de mejorar sus procesos (administrativos, financieros, productivos, etc.) con el fin de hacerse cada vez más fuertes, y poder de este modo ofrecer al mercado productos o servicios de gran calidad, en el momento indicado y a bajo precio, de forma tal que se pueda alcanzar la fidelización de una gran parte del

mercado, el cual permita la sustentabilidad y rentabilidad de la organización ahora y en el futuro.

Una de las áreas donde las organizaciones manufactureras se deben enfocar es en producción. Ello, se debe a que es precisamente ésta área la responsable de la fabricación de los bienes que se harán llegar al mercado, los cuales deben contar con altos niveles de calidad, además de ser entregados en las cantidades oportunas y en el momento indicado por el mercado.

Al interior de esta área, la programación de la producción o Scheduling, juega un papel fundamental, porque es a través de ella como se determina la mejor manera de procesar las ordenes de trabajo, teniendo en cuenta los recursos con los cuales se cuenta (hombres, máquinas, materia primas, etc.), en búsqueda siempre de alcanzar un objetivo (minimizar lapso, tardanza total, tardanza máxima o cualquier combinación entre ellas) por medio del cual se pueda mejorar la productividad y la eficiencia de las organizaciones.

Haciendo una breve reseña histórica del desarrollo del Scheduling, se puede observar como a través del tiempo esta importante labor ha evolucionado y mejorado gracias a modelos y teorías propuestas por grandes investigadores, y por supuesto, por el desarrollo de tecnologías de información. Es así, como debemos comenzar por los diseños realizados por Henry Gantt, quien a principios del siglo XX diseñó sus novedosos diagramas para el control de la producción. Estos diagramas, son considerados como los más antiguos y mejor conocidos gráficos de control, especialmente diseñados para mostrar gráficamente la relación entre el rendimiento planificado y el rendimiento real. Gantt diseñó sus diagramas, para que los supervisores de producción pudieran saber rápidamente si ésta se encontraba en la programación prevista, antes o después de la misma. [2]

Luego del desarrollo de los llamados diagramas de Gantt, se han desarrollado nuevas herramientas que permiten monitorear y planificar la producción, como Loading, Boards and Lines of Balance.

Loading es una técnica de scheduling que asigna una operación a un día o semana específica, en el cual una máquina o conjunto de máquinas, se encargarán de desarrollarla. (MacNiece, 1951). La carga se considera finita cuando tiene en cuenta el número de máquinas, turnos por día, horas de trabajo por día, días por semana, así como el tiempo necesario para completar la orden. [2]

El mismo autor (MacNiece, 1951), también hace referencia a Planning Board, la cual se le atribuye a Frederick Taylor. El tablero, tiene una fila de espacios para cada máquina, y cada fila tiene un espacio para cada turno. Cada espacio contiene una o más tarjetas correspondientes a la orden u órdenes que deberían producirse en ese turno, teniendo en cuenta unas restricciones de capacidad. Una orden de gran tamaño será ubicada en más de un espacio consecutivo. [2]

Las líneas de equilibrio, son utilizadas para determinar hasta qué punto por delante, o por detrás, de un centro de trabajo puede estar la producción de una serie de ensambles idénticos requeridos en el tiempo. Dada la demanda de productos terminados y una lista de materiales con plazos de entrega para completar la fabricación de los componentes y subensambles, se puede calcular el número acumulado de componentes, subensambles y productos terminados que deberían completarse en un instante de tiempo para satisfacer la demanda. Esta herramienta, es utilizada para comparar el número de componentes, subensambles y productos terminados calculados, con el número de componentes, subensambles y productos terminados que actualmente se han hecho en un determinado instante de tiempo.[2]
Luego, en 1956 se da el desarrollo formal de la técnica de CPM (Critical Path Method) por parte de la corporación Du Pont. En 1958, PERT (Program Evaluation

and Review Technique) es creado por la Oficina de Proyectos Especiales de la Marina de Guerra del Departamento de Defensa de los Estados Unidos como parte del proyecto Polaris de misil balístico móvil lanzado desde submarino.

La tecnología de la información, también ha tenido un gran impacto en la forma en la cual se ejecuta la programación de la producción. Entre los muchos avances de ésta área se encuentra la posibilidad de ejecutar algoritmos complejos rápidamente. De este modo, el desarrollo de mejores algoritmos parte fundamental en el desarrollo de la historia de la programación de la producción. [2]

La programación lineal fue desarrollada en 1940´s y aplicada en problemas de planificación de la producción (aunque no directamente a la programación de la producción). En 1947, George Dantzig inventó el método simplex, la cual es una técnica general y extremadamente poderosa para resolver problemas de programación lineal. En la década de los 50´s se llevó a cabo el diseño y desarrollo de algunos importantes algoritmos motivados por problemas de programación de la producción. Algunos de ellos son: La regla de Jhonson para dos máquinas en serie (Flowshop), la regla EDD (The Erliest Due Date) para la minimización de la tardanza máxima, la regla SPT (The Shortest Processing Time) para la minimización del tiempo promedio de flujo [2].

La técnica Branch and Bound aparece alrededor de los 60´s. Este algoritmo, implícitamente enumera todas las posibles soluciones y encuentra una solución óptima. Mientras tanto, otras técnicas como la Relajación de Lagrange, técnica de generación de columnas para programación lineal y programación de restricciones, se han desarrollado para resolver problemas de programación entera [2].

A principios de los 70´s se desarrolla la teoría de la complejidad, la cual muestra porque algunos problemas de programación eran duros. Además de ello, se determina que es poco probable que existan algoritmos que puedan encontrar

soluciones óptimas a estos problemas en cantidades razonables de tiempo. Es así, como a partir de los años 80´s los algoritmos de búsqueda, los cuales pueden entregar soluciones cercanas al óptimo en períodos de tiempo razonables, adquieren gran importancia. Dentro de estos algoritmos de búsqueda se puede mencionar algunos como búsqueda tabú, recocido simulado, colonias de hormigas, algoritmos genéticos, búsquedas en entorno local, y en general una gran cantidad de algoritmos que se han inspirado en teorías biológicas [2].

1.2.2. PROBLEMA.

El medio actual en el que se desenvuelven las organizaciones es altamente dinámico y competitivo, por lo tanto estas deben responder cada vez más eficiente y rápido a los cambios del entorno y las exigencias de los clientes, pero en muchas empresas la eficiencia y la capacidad de respuesta no es la adecuada para el mercado en el cual se encuentran. El área de producción como parte vital de las empresas permite medir la capacidad de estas, su efectividad, y si se administra correctamente agiliza la habilidad de respuesta de las organizaciones ante las exigencias de la demanda, punto importante para sobrevivir y destacar en el medio, brindando más y mejor satisfacción a sus clientes.

Uno de los aspectos que más influyen en la organización de una empresa es la programación de la producción, a la hora de programar se enfrentan a la decisión de establecer la secuencia de procesamiento óptima de las órdenes de pedido. En la mayoría de los casos la programación de las órdenes a las diferentes estaciones de trabajo (scheduling) se realiza de una manera manual consiguiendo con esto una solución alejada de lo que llamamos el óptimo o una solución eficiente, ocasionando retrasos en entregas, uso de recursos excesivo e inventarios elevados y con ello pérdida de competitividad y productividad y en el peor de los casos desaparición de la organización.

1.2.3. SÍNTOMAS DEL PROBLEMA.

En general, cuando una organización está realizando mal la programación de producción, ello se ve reflejado en un uso desmedido de recursos, hay un bajo índice de satisfacción de los clientes, dado que sus pedidos generalmente no son entregados ni en las cantidades ni en el momento adecuado.

Además de ello, la organización se ve obligada a pagar altas multas por el incumplimiento en dichas fechas de entrega; la productividad es baja, los clientes de la organización comienzan a buscar nuevas y mejores organizaciones, las cuales si puedan satisfacer sus necesidades de manera eficiente, lo que conlleva a perder participación en el mercado; la competitividad de la organización también es deficiente, y los niveles de rentabilidad comienzan a descender como consecuencia de ello.

1.2.4. CAUSAS DEL PROBLEMA

Son muchos los factores que pueden influir dentro de las organizaciones para que estas no puedan desarrollar de manera eficiente la programación de la producción. Sin embargo, es posible destacar principalmente las siguientes:

- La labor de programar la producción es muy compleja, debido a que los métodos habitualmente utilizados no proporcionan resultados óptimos en períodos de tiempo relativamente cortos, e incluso, en muchas ocasiones dichos métodos no son capaces de alcanzar soluciones medianamente cercanas al óptimo en periodos razonables de tiempo.

- La implementación de herramientas tecnológicas que simplifiquen y faciliten la determinación de un programa eficiente de producción es sumamente costoso; por lo cual solo aquellas organizaciones que cuenten con capital suficiente para mantenerse a la vanguardia de los avances de la ciencia y la tecnología, podrán hacerse de dichas herramientas.

- Existencia de un canal de comunicación ineficiente entre las áreas de venta y producción, las cuales deben mantener un intercambio constante de información, en pro de poder definir mecanismos y estrategias idóneas para poder satisfacer las necesidades de todos los clientes de la organización, manteniendo así la competitividad de la misma.

El entorno en el cual se desenvuelven las organizaciones hoy día es sumamente dinámico, cada vez hay menos contacto directo con los clientes, y estos a su vez se hacen más exigentes; todo ello, en conjunto, hace mucho más compleja la labor de las organizaciones.

1.2.5. PRONÓSTICO

Las organizaciones hoy día deben estar siempre enfocadas a alcanzar altos niveles de competitividad y rentabilidad; para ello, se hace imperioso que éstas emprendan acciones y estrategias encaminadas a alcanzar dichos objetivos, de lo contrario, su improductividad, ineficacia e ineficiencia, las llevarán a una profunda crisis en donde sus debilidades quedarán expuestas, perdiendo, de esta manera, terreno en el mercado en el cual compiten, produciendo ello que aquellas que estén mucho más preparadas ocupen su lugar y terminen sacándola por completo del mercado mundial en el cual se compite hoy día.

1.2.6. CONTROL AL PRONÓSTICO

Como ya se mencionó antes, la labor de la programación de la producción es indispensable para que las organizaciones puedan ser más productivas y eficientes. Por otro lado, no debemos olvidar que no solo ésta área es responsable de la competitividad de las organizaciones, por que éstas se comportan como sistemas, y como tales, las partes que la componen deben actuar de manera conjunta para alcanzar los objetivos deseados. Teniendo en cuenta esto, por

medio de la presente investigación, se pretende desarrollar una herramienta informática que permita desarrollar de manera más eficiente la programación de las actividades productivas, permitiendo, la satisfacción de los requerimientos del cliente en cuanto a la calidad de los bienes requeridos, las cantidades deseadas y los tiempos de entrega estipulados; contribuyendo así, a la mejora de la competitividad de las organizaciones, lo cual hace parte fundamental de los objetivos que se pretenden alcanzar en cualquiera de ellas.

1.2.7. INTERROGANTES Y FORMULACIÓN DEL PROBLEMA

Los anteriores planteamientos, permiten formular el siguiente interrogante: ¿la metaheuristica viral systems se puede aplicar con buenos resultados a la programación de la producción de sistemas flexible flow shop?

De esta pregunta general se desprenden otras que permite desglosar el problema, tales como: ¿Cuál es el modelo matemático de programación de la producción de sistemas flexible flow shop?, ¿Cuáles son los algoritmos más utilizados para la programación de la producción de sistemas flexible flow shop?, ¿qué tan buena es la metaheuristica viral systems desde el punto de vista de la calidad de la respuesta y la eficiencia computacional para solucionar el problema de programación de la producción de sistemas flexible flow shop, comparándola con otros algoritmos existentes?.

1.3. OBJETIVOS

1.3.1. OBJETIVO GENERAL

Adaptar y aplicar la metaheurística Viral Systems a problemas de programación de la producción de sistemas Flexible Flowshop y determinar la eficiencia de los resultados arrojados por la misma, comparándolos con los resultados arrojados por otras metaheurísticas encontradas en la literatura, que han sido aplicadas para tratar de dar solución a este tipo de problemas.

1.3.2. OBJETIVOS ESPECÍFICOS

- Adaptar la metaheurística Viral Systems para solucionar problemas de programación de la producción en sistemas Flexible Flowshop.

- Comparar la eficiencia de los resultados obtenidos al aplicar la metaheurística Viral Systems en problemas de programación de la producción de sistemas flexible Flowshop, con los resultados arrojados por otras metaheurísticas que se han aplicado a este mismo tipo de problemas.

- Identificar ventajas y desventajas que presenta la metaheurística Viral systems en comparación con otras metaheurísticas utilizadas en este mismo campo.

1.4. JUSTIFICACIÓN

En el mundo actual globalizado, altamente competitivo y dinámico, las organizaciones requieren ser cada más eficientes, eficaces y oportunos, es por esto que necesitan contar con herramientas tecnológicas que le permitan administrar mejor su quehacer, conseguir ventajas competitivas, mejorar la productividad o contrarrestar efectos del entorno (competidores, proveedores, gobierno, ambiente, etc). Unas de las áreas que más requiere contar con tecnologías cada vez más efectivas es producción, en la cual una mala programación de la producción puede ocasionar la pérdida de un negocio, de clientes, de ingresos o un aumento de costos y con ello disminución en la rentabilidad y competitividad de la empresa.

De esta coyuntura se desprende la necesidad de las organizaciones de contar con sistemas informáticos que basados en algoritmos resuelvan en tiempos relativamente cortos la programación de la producción de la empresa, generando soluciones muy buenas desde el punto de vista de la eficiencia y eficacia, ayudando con ello a la supervivencia y desarrollo de la organización en un entorno cada vez más dinámico y complejo.

Como se ha dicho anteriormente, el área de producción cumple con una labor importante dentro de toda organización, pues de ella depende que los clientes puedan recibir de manera oportuna los bienes requeridos en las cantidades, momento y lugar indicado. Sin embargo, la complejidad de esta labor muchas veces atenta contra el cumplimiento de esta premisa.

Es por ello, que se hace necesario que toda organización cuente, dentro de su área de producción, con las herramientas necesarias, que le permitan realizar la labor de programación de la producción de la manera más eficiente posible, evitando así, problemas como el uso inadecuado de recursos de producción,

entregas tardías, pago de multa por retrasos, y en general, evitar situaciones que impidan mejorar la productividad y la rentabilidad de toda la organización. Decisiones como la liberación de órdenes de producción, asignación de recursos (personas, equipos, herramientas, etc.) a tareas, reasignación de recursos de una tarea a otra, priorización de las ordenes que requieren los mismo recursos, secuenciación de tareas, determinación del momento en que deben iniciar y finalizar las tareas e interrupción de las mismas que debían ser detenidas son claves en cualquier sistema de programación de la producción [3], y por ello deben desarrollarse herramientas que faciliten dicha labor.

Sin embargo, hablar de programación de la producción en términos generales, abarca una gran cantidad de modelos susceptibles de ser programados. Este trabajo, se enfocará en la programación de la producción en sistemas Flexible Flowshop, donde las órdenes de producción deben pasar en un mismo orden por las k estaciones del sistema, cada una de las cuales puede tener dos o más máquinas en paralelo. Lo que se busca con una eficiente programación de la producción, es alcanzar uno o varios objetivos (minimización del lapso, tardanza máxima, tardanza total), siendo estos últimos, los más comunes en la cotidianidad de las organizaciones.

Por ello, resulta sumamente interesante, observar y determinar la manera en la cual la metaheurística Viral Systems puede ayudar a solucionar este tipo de problemas, debido a que los algoritmos exactos que se conocen, muchas veces no arrojan las soluciones en tiempos razonables, o en muchos casos, ni siquiera pueden calcular un valor óptimo; y las heurísticas, no dan certeza de que tan cercana al óptimo es la solución encontrada.

Por otro lado, se ha tomado la decisión de aplicar la metaheurística Viral Systems a este tipo de problemas, porque ésta ha sido desarrollada muy recientemente, y no se han encontrado fuentes bibliográficas que demuestren su utilización para resolver problemas de programación de la producción de sistemas Flexible

Flowshop, por lo cual se hace interesante determinar los resultados arrojados por este algoritmo, para luego, compararlos con otras metaheurísticas encontradas en la literatura, que ya han sido implementadas en este tipo de problemas.

1.5. ALCANCES Y LIMITACIONES

La metaheurística Viral Systems se empleará en la programación de la producción de sistemas Flexible Flowshop con máquinas paralelas no relacionadas en cada estación sujeta a restricciones de tiempos de liberación y tiempos de preparación dependientes de la secuencia o de las máquinas, cuyo objetivo principal es la minimización del Makespan (Cmax).

CAPÍTULO 2 : MARCO DE REFERENCIA TEÓRICO GENERAL

2.1. MARCO TEÓRICO

2.1.1. PLANEACIÓN DE LA PRODUCCIÓN

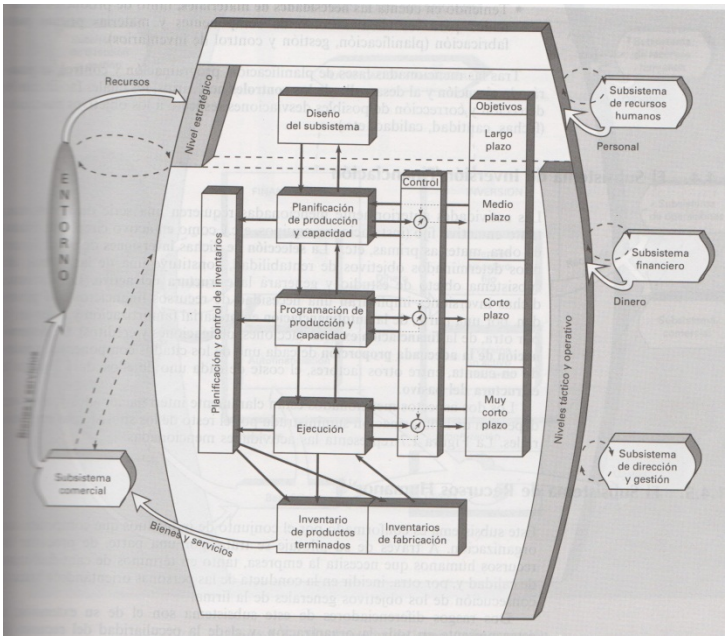
El proceso de dirección y gestión de operaciones comienza definiendo los objetivos a largo plazo, acordes con los globales de la empresa, y diseñando estrategias coherentes con los mismos, estos objetivos y estrategias deben presidir el diseño del subsistema, proceso en el cual se deciden inversiones en estructura, teniendo gran importancia los criterios tecnológicos, económicos financieros, así como otros menos cuantificables.[4]

Se dispone de un marco de referencia que indica las metas a conseguir a largo plazo, cómo y con qué medios. A partir de este hay que descender al medio y corto plazo a través de distintas actividades interrelacionadas:

- Concretar los objetivos del subsistema de operaciones acordes con los objetivos y metas de la empresa.
- Determinar las actividades de productos y servicios a elaborar, así como los correspondientes momentos de tiempo (Planeación agregada).
- Decidir que subconjuntos y componentes hay que producir o adquirir y en que fechas, para satisfacer el plan elaborado para los productos, esta planeación es a mediano plazo (Programa Maestro de Producción).
- Tener en cuenta las necesidades de materiales, tanto de productos terminados para los clientes como de componentes y materias primas para la fabricación (planificación, gestión y control de inventarios).
- Programar actividades que deberán desarrollar las distintas unidades productivas, y en qué momentos, para cumplir lo previsto en la fase anterior (Programación producción).

- Considerar en todos los niveles la problemática de la capacidad, de forma que se elaboren planes y programas factibles.[4]

Ilustración 1: Subsistema de Operaciones



Fuente: Domínguez Machuca. Et al, Dirección de operaciones

2.1.2. PROBLEMA DE PROGRAMACIÓN DE LA PRODUCCIÓN (PRODUCTION SCHEDULING)

La programación de la producción ha sido estudiada muy ampliamente en las últimas cinco décadas, por la importancia que tiene en las organizaciones. Según Pinedo[5] Programación de la producción trata de la asignación de recursos a tareas durante periodos de tiempo, cuya meta es optimizar uno o varios objetivos, tanto los recursos como las tareas pueden tomar muchas formas dependiendo de la organización, por ejemplo los recursos pueden ser maquinas en un taller, pistas de aterrizaje en un aeropuerto, equipos en una construcción, unidades de procesamiento en un ambiente de computación, etcétera. En cuanto a las tareas

pueden ser operaciones en un proceso de producción, el abordaje y los aterrizajes en un aeropuerto, etapas en un proyecto de construcción, ejecución de programas de computación, etcétera.

Por su parte Wight [2] define scheduling como el establecimiento del tiempo en la cual debe realizarse una tarea, y detalla que en una empresa existen múltiples tipos de scheduling en las empresas incluyendo la programación de una orden en un taller, en la cual se muestra cuando debe iniciar y finalizar cada operación.

Mientras que Cox [2] considera scheduling como el establecimiento de fechas de inicio y/o finalización a las operaciones o grupos de operaciones, que muestran cuando deben ser realizadas las ordenes de producción para ser completadas a tiempo.

Baker [6] define la programación de la producción como la asignación de recursos disponibles de producción en el tiempo para fabricar un conjunto de tareas.

En general se puede definir programación de la producción como la asignación de los recursos disponibles a un conjunto de tareas u órdenes de fabricación en un horizonte de tiempo, buscando que las salidas se obtengan en los tiempos deseados.

2.1.2.1. NOTACIÓN

Para el presente documento se utilizará la notación sugerida por Pinedo [5]

n : Número de trabajos a procesar
m : Número de maquinas
$p_{i\,j}$: Tiempo de proceso del trabajo j en maquina i

r_j : Tiempo de liberación de la orden de trabajo j. es decir, el tiempo en que la orden entra al sistema, también se puede considerar como el tiempo más temprano que el trabajo j puede ser procesado.

d_j : Fecha de entrega del trabajo j. Si el trabajo j es terminado después de esta fecha es aceptada pero incurre una penalización. Si se debe considerar como una fecha última de entrega debe ser conocido como deadline y se simboliza como $\bar{d}_j$.

w_j : Importancia relativo del trabajo j

Un problema de scheduling se describe como una tripleta α | β | γ. El campo α describe la configuración del sistema y contiene solo una entrada. El campo β proporciona detalles sobre las características de procesamiento del taller y las restricciones, este campo puede no contener entradas, o tener una sola o múltiples entradas. El campo γ describe el objetivo a optimizar.

Notación matemática para el campo α:

- 1 : 1 máquina
- Pm : Máquinas idénticas en paralelo
- Qm : Máquinas en paralelo con velocidades diferentes
- Rm: Máquinas en paralelo no relacionadas.
- Fm : m máquinas en serie
- FFc : Flowshop flexible (híbrido) con c estaciones en serie
- Jm : Jobshop con m máquinas
- FJc : Jobshop flexible (híbrido) con c estaciones de m máquinas idénticas
- Om: OpenShop aquí hay m máquinas. No hay ninguna restricción con respecto a direccionamiento de cada trabajo a través del sistema.

Notación matemática para el campo β

- r_j : Tiempo de liberación de la orden del trabajo j, si este símbolo se muestra en el campo β entonces el trabajo j no puede iniciar antes del tiempo rj. Es el tiempo transcurrido entre la finalización de la orden j y el inicio del trabajo k.

- S_{jk} : Tiempos de preparación dependientes de la secuencia, es el tiempo entre la finalización del trabajo j y el inicio del trabajo k. si el tiempo de preparación entre el trabajo j y la orden k dependen de la maquina entonces se agrega el sufijo i por ejemplo Sijk.

- prpm : Preemption, existe la posibilidad de culminar la operación sobre un trabajo (orden) en diferentes máquinas, en otras palabras no es necesario que una vez iniciado un trabajo en una máquina, se finalice en la misma.

- prec : Restricciones de precedencia, esto implica que uno o varios trabajos deben finalizarse antes que otro trabajo(orden) inicie.

- brkdwn: Fallas en los recursos, esto implica que las maquinas no están disponibles continuamente.

- M_j : Elegibilidad de recursos (para problemas Pm), esto implica que no todas las m maquinas son capaces de fabricar el trabajo j, entonces el conjunto Mj establece las máquinas que son capaces de fabricar el trabajo j.

- prmu : Programa de Permutación (secuencia se mantiene), aparece en ambientes flowshop en la cual las órdenes son programadas para un recurso y luego se extiende para todos los demás recursos, es decir se mantiene en todo el sistema la disciplina FIFO (primero en entrar primero en salir).

- block : Bloqueos es un fenómeno que ocurre en ambientes flowshop, en la cual existen buffer limitados entre dos estaciones consecutivas, entonces podría pasar que cuando el buffer está lleno, la máquina que se encuentra primera en la secuencia no puede entregar el trabajo que ha finalizado hasta que la otra máquina finalice el trabajo que está realizando, de esta forma se evita que la primera máquina continúe la fabricación de la siguiente orden o producto.

- nwt : No-wait, es otro fenómeno que ocurre en flowshop, en la cual los trabajos no pueden esperar entre dos máquinas sucesivas, esto implica que un trabajo que puede ser procesado en una maquina se fabrique sin tener que esperar otra máquina.

- recrc : Recirculación de trabajos, puede ocurrir en job shop y flexible job shop, en la cual un trabajo puede visitar una máquina o un centro de trabajo más de una vez.

- Fmls: Job families, Los n trabajos pertenecen en este caso a F diferentes familias de trabajo. Los trabajos de la misma familia pueden tener tiempos de procesamiento diferentes, pero pueden ser procesados en una máquina uno después de otro sin requerir alistamiento entre ellos. En cambio si la máquina cambia de fabricar la familia g a la familia h entonces se requiere alistamiento.

- batch(b): Batch processing, una máquina o estación de trabajo está disponible para fabricar un conjunto b de de trabajos simultáneamente.

- Cualquier entrada adicional en el campo β debe ser auto-explicatoria, ejemplo pj=p implica que todos los tiempos de procesamiento son iguales

Notación matemática para el campo γ

- C_j: Tiempo de finalización de la orden j

- L_j: Latencia del trabajo j $L_j=C_j-d_j$. la cual es positiva si el trabajo fue finalizado después del tiempo de prometido de entrega, y negativo si el trabajo fue finalizado antes del due date.

- T_j: Tardanza del trabajo j. $T_j=\max(C_j-d_j , 0)=\max(L_j ,0)$.

- U_j: Penalización unidad, si Cj > dj , entonces Uj=1; y Uj=0, en cualquier otro caso.

- Cmax: Makespan (lapso): es el tiempo de finalización del último trabajo en el sistema. Cmax= max(C1, …, Cn).

- Lmax: Latencia máxima. Lmax= max(L1, …, Ln). Mide la peor infracción de las fechas de vencimiento (duo dates).

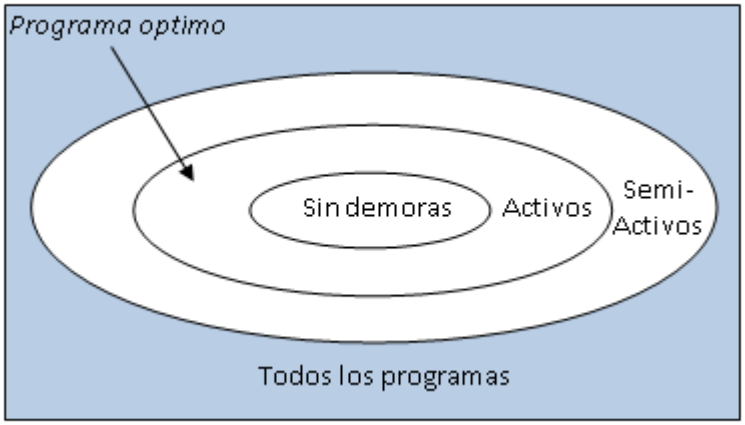
- E_j: Adelanto del trabajo j. $E_j=\max(d_j -C_j , 0)=\max(-L_j ,0)$.

2.1.2.2. CLASES DE PROGRAMAS

- Sin retraso (nondelay): un programa factible es llamado nondelay cuando ninguna máquina se mantiene desocupada al tener trabajos para ser procesados.
- Activo: un programa es llamado activo si no es posible obtener otro programa por intercambios de trabajos sin perjudicar la factibilidad del programa.

- Semi-activo: un programa es llamado semi- activo si ninguna tarea puede ser terminada más temprano sin afectar el orden de procesamiento en cualquiera de las máquinas.

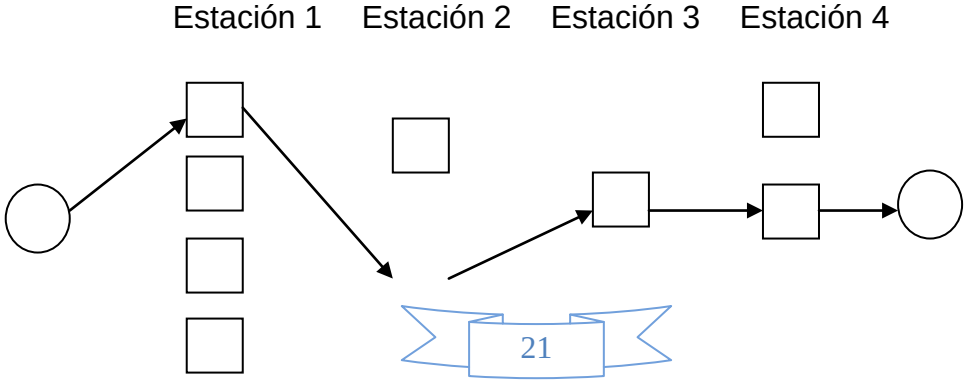
Ilustración 2: Diagrama de Venn de las clases de programas



2.1.3. EL PROBLEMA DE PROGRAMACIÓN DE LA PRODUCCIÓN EN SISTEMAS FLEXIBLE FLOWSHOP (FFS)

Los flexible flowshop (hibrid flow shop) son ambientes comunes en la manufactura, en la cual un conjunto de n trabajos son procesados en series (siguen la misma secuencia de operación) en m estaciones de trabajo con m ≥ 2 , cada una de las k estaciones tiene $M^{(k)}$ maquinas en paralelo con $M^{(k)} \geq 1$ y al menos un $M^{(k)}>1$ (ver [7].). La ilustración 2 muestra un ejemplo del flujo de un trabajo en un FFS con cuatro estaciones de trabajo.

Ilustración 3: Flujo de Trabajo en un sistema FFS

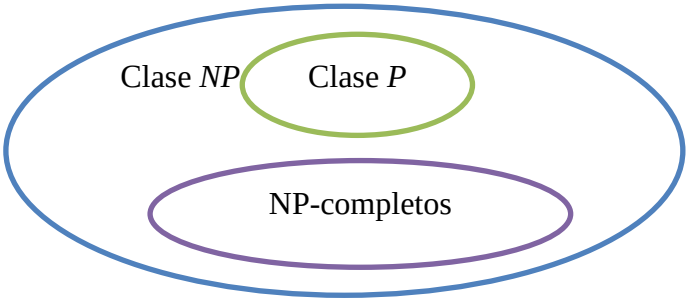


Los sistemas flexible flowshop son encontrados en muchas clases de escenarios reales como son: la fabricación de elementos electrónicos, papeles y textiles, producción de concreto, se han encontrado ejemplos en ambientes diferentes a la manufactura, como la arquitectura de servicios de internet, sistemas de manejo de contenedores, entre otros [7].

2.1.3.1. COMPLEJIDAD

Un aspecto importante de la teoría computacional es categorizar los problemas según su complejidad. Una clase de complejidad representa el conjunto de todos los problemas que pueden resolverse con una determinada cantidad de recursos computacionales. Hay dos clases importantes de problemas: *P* y *NP*.

Ilustración 4: Clasificación de problemas según su complejidad



La clase de complejidad *P* representa el conjunto de los problemas que pueden ser resueltos por un algoritmo determinístico en un tiempo polinómico. Un algoritmo (determinístico) es polinómico de un problema si su más alta complejidad está limitada por una función polinómica $p(n)$ donde n representa el tamaño de la instancia de entrada I. Por lo tanto, la clase P representa a la familia de los problemas que pueden ser resueltos por algoritmos de tiempo polinómico.

Los problemas que pertenecen a la clase P son entonces relativamente "fáciles" de resolver.

La clase de complejidad *NP* es el conjunto de los problemas que pueden ser resueltos por un algoritmo no determinístico en un tiempo polinómico. Un algoritmo no determinístico contiene uno o más puntos de elección en los que varios "caminos" diferentes son posibles sin ningún tipo de especificación, de los cuales uno será tomado.

La cuestión de si $P = NP$[15] es una cuestión de mucha importancia por la gran repercusión que la respuesta tendría en la teoría de complejidad computacional. Obviamente, para cada problema en *P* tenemos un algoritmo no determinístico para resolverlo. Entonces, $P \subseteq NP$. Sin embargo, la conjetura $P \subset NP$ es todavía una cuestión abierta.

Un problema *A* se reduce a un problema *B* polinómico si, por todas las instancias de entrada de I_A para *A*, siempre se puede construir una instancia de entrada para I_B para *B* en una función polinómica de tamaño $L(I_A)$ de la entrada de I_A, tales que I_A es una instancia positiva de *A* si y sólo si I_B es una instancia positiva de *B* [32].

Un problema $A \in NP$ es *NP-completo* si todos los demás problemas de la clase *NP* se reducen al problema polinómico A. La figura muestra la relación entre *P*, *NP*, y los problemas *NP-completos*. Si un algoritmo determinístico polinómico existe para resolver un problema *NP-completo*, entonces todos los problemas de la clase *NP* puede ser resueltos en tiempo polinómico.
Los problemas *NP-HARD* son problemas de optimización cuyas decisiones son problemas *NP-completos*. La mayoría de los problemas de optimización del mundo real son *NP-HARD* para los que probadamente no existen algoritmos eficaces. Requieren tiempo exponencial (a menos que *P=NP*) para resolver en la optimalidad. Las metaheurísticas constituyen una importante alternativa para resolver esta clase de problemas.

Los problemas FFS son en la mayoría de los casos Np- Hard, para la instancia de dos estaciones, en las cuales una estación tiene dos máquinas en paralelo y la

otra una máquina, es NP-hard, después de los resultados de Gupta [8]. De igual forma los FFS cuando se admite preemption son Np-hard en sentido fuerte con dos máquinas [9]. Sin embargo con algunas propiedades especiales y relaciones de precedencia, el problema podría ser polinomialmente solucionable [10].

2.1.3.2. MODELACIÓN MATEMÁTICA

Un modelo de programación entera mixta interesante fue propuesto por Jungwattanakit [6] el cual se seguirá en el presente documento.

El modelo matemático se basa en los siguientes supuestos:

- Se tiene un conjunto n independiente de órdenes de compra. j ϵ {1, 2…,n}
- Cada orden de compra tiene asociado un tiempo de entrega d_j
- Las órdenes de compra no arriban simultáneamente, por lo tanto estas disponibles tiempos de liberación r_j.
- Cada trabajo j consiste en una cadena de k (k≥2) operaciones diferentes simbolizadas como O_j^1, …, O_j^k. Las cuales deben ser procesadas en este orden.
- Cada trabajo j tiene un tiempo estándar de procesamiento fijo, simbolizado como ps_j^t en cada estación t. t ϵ {1, 2…,k}.
- En cada estación t hay un conjunto m^t de maquinas paralelas no relacionadas. i ϵ {1, 2…,m^t}, pueden existir estaciones con solo una maquina, pero al menos una debe tener múltiples maquinas.
- La máquina i de la estación t puede procesar la orden j con una velocidad relativa V_{ij}^t.
- El tiempo de procesamiento p_{ij}^t de la orden j en la maquina i de la estación t es igual a ps_j^t / V_{ij}^t.
- Este problema asume un ambiente estático y determinístico.

- No hay relaciones de precedencia entre los trabajos.
- No se admite Preemption, es decir una orden que es iniciada debe finalizar en la misma máquina sin interrupción.
- Las operaciones de un trabajo se deben realizar secuencialmente, y no se puede procesar la misma orden en dos estaciones diferentes al mismo tiempo.
- Ningún trabajo puede ser procesado por más de una máquina al mismo tiempo
- Ninguna maquina puede procesar más de una orden al mismo tiempo.
- No se permite el fraccionamiento de las órdenes de trabajo.
- Los tiempos de alistamiento para este problema son considerados de dos tipos: (1) tiempo de alistamiento dependiente de la máquina, esto ocurre únicamente cuando el trabajo es asignado en la maquina en la primera posición se denota como ch_{ij}^{t} si el trabajo j es asignado a la maquina i de la estación t en la primera posición y (2) tiempo de alistamiento dependiente de la secuencia. Se simboliza como s_{lj}^{t}, es el tiempo de alistamiento entre la orden l y la orden j en la estación t. aquí el trabajo l es procesado antes del trabajo j en la misma máquina.
- Todos los tiempos de alistamiento son conocidos y constantes.

Notaciones:

- t índice de la estación, t= 1, 2, 3, …, k
- i índice de la máquina, i= 1, 2, 3, …, m^t
- j, l índice de órdenes, j,l= 1, 2, 3, …, n
- r_j fecha de lanzamiento de la orden j.
- m^t número de máquinas en paralelo de la estación t
- d_j fecha de vencimiento del trabajo j.
- s_{jl}^{t} tiempo de alistamiento entre la orden j y la orden l en la estación t

- ch_{ij}^{t} tiempo de alistamiento de la orden j si esta es asignada a la maquina i en la primera posición de la estación t.
- ps_{j}^{t} tiempo estándar de procesamiento de la orden j en la estación t.
- v_{ij}^{t} velocidad relativa de la maquina i de la estación t para el trabajo j.
- a_{i}^{t} tiempo cuando la maquina i de la estación t se pone disponible.
- X_{ijl}^{t} 1 si la orden j es programada inmediatamente antes que la orden l en la maquina i de la estación t, y 0 es otro caso.
- O_{j}^{t} tiempo de operación de la orden j en la estación t.
- C_{j}^{t} tiempo de finalización de la orden j en la estación t.
- C_{max} el makespan (lapso).
- U_{j} una variable booleana; 1 si la orden j es finalizada tarde, y 0 en otro caso.
- T_{j} tardanza de la orden j
- η_{T} numero total de trabajos tardíos en el programa.

El problema puede ser formulado como sigue:

Ecuación 1: Modelación Matemática Flexible FlowShop

$$minimizar\ \lambda C_{max} + (1-\lambda)\eta_{T} \qquad (1)$$

$$sujeto\ a:$$

$$\sum_{i=1}^{m^t}\sum_{j=0}^{n} X_{ijl}^{t} = 1, \qquad \forall t, l\ (2)$$

$$\sum_{i=1}^{m^t}\sum_{l=1}^{n+1} X_{ijl}^{t} = 1, \qquad \forall t, j\ (3)$$

$$\sum_{l=1}^{n+1} X_{i0l}^{t} = 1, \qquad \forall t, i\ (4)$$

$$\sum_{j=0}^{n} X_{ij(n+1)}^{t} = 1, \qquad \forall t, i\ (5)$$

$$X_{ijj}^{t} = 0, \qquad \forall t, i, j\ (6)$$

$$\sum_{j=0}^{n} X_{ijl}^{t} = \sum_{j=1}^{n+1} X_{lij}^{t}, \qquad \forall\, t,i,l\ (7)$$

$$X_{ijl}^{t} \in \{0,1\}, \qquad \forall\, t,i,j,l; j=0; l=n+1\ (8)$$

$$O_j^t = \sum_{i=1}^{m^t}\sum_{l=1}^{n+1} \frac{ps_j^t}{v_{ij}^t} X_{ijl}^t, \qquad \forall\, t,j\ (9)$$

$$C_l^t - C_j^t \geq S_{jl}^t + O_l^t + \left(\left(\sum_{i=1}^{m^t} X_{ijl}^t\right) - 1\right) B, \qquad \forall\, t,j,l; j \neq l\ (10)$$

$$C_j^t \geq 0, \qquad \forall\, t,j\ (11)$$

$$C_l^t - C_l^{t-1} \geq \sum_{i=1}^{m^t}\sum_{j=1}^{n} X_{ijl}^t S_{jl}^t + \sum_{i=1}^{m^t} ch_{il}^t X_{i0l}^t + O_l^t, \qquad \forall\, t,l\ (12)$$

$$C_j^0 \geq \eta_j, \qquad \forall j\ (13)$$

$$C_j^t \geq \sum_{i=1}^{m^t} a_i^t X_{i0j}^t + \sum_{i=1}^{m^t} ch_{il}^t X_{i0l}^t + O_j^t, \qquad \forall t,j\ (14)$$

$$C_{max} \geq C_j^k, \qquad \forall j\ (15)$$

$$T_j \geq C_j^k - d_j, \qquad \forall j\ (16)$$

$$T_j \geq 0, \qquad \forall j\ (17)$$

$$U_j \leq BT_j, \qquad \forall j\ (18)$$

$$BU_j \geq T_j, \qquad \forall j\ (19)$$

$$\eta_T = \sum_{j=1}^{n} U_j, \qquad \forall j\ (20)$$

$$U_j \in \{0,1\}, \qquad \forall j\ (21)$$

- La ecuación (1) describe la función objetivo. Si tenemos X_{i0l}^t =1 describe que la orden l es secuenciada como la primera orden en la maquina i de la estación t.
- Las restricciones (2) – (8) aseguran que el programa parcial en cada una de las máquinas de cada una de las estaciones sean factibles.
- El conjunto de restricciones (2) y (3) aseguran que solamente un trabajo es asignado a cada una de las posiciones de la secuencia en cada una de las estaciones.

- El conjunto de restricciones (4) y (5) asegura que solamente un trabajo sea asignado a la primera y última posición en cada máquina en cada estación.
- La restricción (6) garantiza que después de la orden ha sido finalizado en cualquier estación, no sea reprocesada en la misma estación.
- La restricción (7) fuerza a construir una secuencia consistente en cada estación.
- La restricción (8) especifica la variable X_{ijl}^{t} como binaria.
- La restricción (9) determina el tiempo de operación de cada trabajo la cual es dependiente de la máquina.
- Las restricciones (10)-(14) encuentran el tiempo de finalización de cada trabajo.
- La restricción (10) es un conjunto de restricciones disyuntivas, consideremos una estación en particular, si las ordenes j y l son programadas en la misma máquina, donde el trabajo j es programado antes que el trabajo l, entonces el trabajo j deberá finalizar su procesamiento antes que el trabajo l pueda iniciar. Este conjunto de restricciones fuerzan a la orden l a seguir la orden j por al menos el tiempo de procesamiento del trabajo l más tiempo de alistamiento de j a l, si el trabajo l es secuenciado inmediatamente después del trabajo j. El valor de B es establecido en una constante muy grande (más grande que la suma de todos los tiempos de procesamiento y alistamiento de los trabajos).
- La restricción (11) asegura que el tiempo de finalización de cada trabajo en cada estación es un valor no negativo.
- La restricción (12) especifica el conjunto de restricciones de precedencia para los trabajos, la cual dice que un trabajo no puede iniciar su procesamiento en la estación t+1antes de haber finalizado en la estación t.
- La restricción (13) aplica solo a la primera estación, diciendo que un trabajo no puede iniciar su procesamiento en la estación uno antes de su fecha de liberación.
- La restricción (14) aplica solamente a los trabajos que fueron asignados como la primera secuencia en cada una de las máquinas, esto es que la

orden no puede iniciar su procesamiento hasta que la máquina esté disponible.

- El conjunto de restricciones (15) enlaza con la variable de decisión makespan.
- El conjunto de decisiones (16) y (17) determina el valor correcto de la tardanza.
- El conjunto de restricciones (18)-(21) enlaza con la variable de decisión número de trabajos tardíos.

2.1.4. ALGORITMOS DE SOLUCIÓN DE PROBLEMAS DE PROGRAMACIÓN DE LA PRODUCCIÓN.

2.1.4.1. MÉTODOS EXACTOS

En la clase de métodos exactos se pueden encontrar los siguientes algoritmos clásicos: la programación dinámica, la familia de algoritmos branch y X (branch and bound, branch and cut, branch and price) desarrollados en la comunidad de investigación de operaciones, la programación con restricciones, y la familia A * de algoritmos de búsqueda (A*, la profundización de la IDA *- algoritmos iterativos) [11] desarrollado en la comunidad de inteligencia artificial [12]. Los métodos enumerativos pueden considerarse como algoritmos de búsqueda de árboles, en donde la búsqueda se realiza a través del espacio de búsqueda de interés general, subdividiendo el problema en problemas más simples.

La programación dinámica se basa en la división recursiva de un problema en subproblemas más simples. Este procedimiento se basa en el principio de Bellman, que dice que *"la subpolítica de una política óptima en sí es óptima"* [13]. Esta etapa del método de optimización es el resultado de una secuencia de decisiones parciales. El procedimiento evita una enumeración total del espacio de búsqueda mediante la poda de secuencias parciales, decisión que no puede conducir a la solución óptima.

Los algoritmos *branch and bound* y *A* se basan en una enumeración implícita de todas las posibles soluciones del problema de optimización tenido en cuenta. El espacio de búsqueda se explora de forma dinámica con la construcción de un árbol cuyo nodo raíz representa el problema que se resuelve todo y su espacio de búsqueda asociado. Las hojas son las posibles soluciones y los nodos internos son subproblemas del espacio de soluciones totales. La poda del árbol de búsqueda se basa en una función de límite que poda subárboles que no contienen ninguna solución óptima [14].

La programación con restricciones es un lenguaje construido en torno a los conceptos de búsqueda en un árbol y sus consecuencias lógicas. Los problemas de optimización en la programación con restricciones se modelan a través de un conjunto de variables ligadas por un conjunto de restricciones. Las variables toman sus valores en un dominio finito de enteros. Las limitaciones pueden tener formas matemática o simbólica[14]

Los métodos exactos se pueden aplicar a pequeñas instancias de problemas difíciles. En la Tabla No. 1 se presentan algunos problemas de optimización *NP-HARD* populares, y el tamaño máximo de los casos en los que los métodos exactos pueden encontrar soluciones de manera óptima. Algunos de los algoritmos exactos usados se implementan en grandes redes de estaciones de trabajo (plataformas de computación en red), integrado por más de 2000 procesadores con más de 2 meses de tiempo de cálculo [15].

Tabla 1: Casos de problemas HP-HARD que se pueden resolver por métodos exactos

Problema de optimización	Asignación cuadrática	Flow-shop *Scheduling*	Coloración grafica	Capacidad de ruteo de vehículos
Tamaño de la instancia	30 objetos	100 trabajos, 20 maquinas	100 nodos	60 clientes

El tamaño de la instancia no es el único indicador que describe la dificultad de un problema, sino también su estructura. Para un problema dado, algunas instancias pequeñas no pueden ser resueltas por un algoritmo exacto, mientras que algunas instancias de gran tamaño pueden ser resueltas exactamente por el mismo

algoritmo. La Tabla No. 2 muestra algunos problemas de optimización populares, (por ejemplo, SOP: problema secuencial de pedido; QAP: problema de asignación cuadrática; GC: coloración gráfica) instancias pequeñas que no se resuelven con exactitud y grandes instancias resueltas por la optimización exacta [14].

Tabla 2: Algunos problemas de optimización pequeños sin resolver de manera exactas y grandes instancias que si se pudieron resolver de manera exacta

Problema de optimización	SOP	QAP	GC
Tamaño de instancias no resueltas	53	30	125
Tamaño de instancias resueltas	70	36	561

2.1.4.2. MÉTODOS APROXIMADOS

En la clasificación de métodos aproximados, se pueden distinguir dos subclases de algoritmos: los algoritmos de aproximación y algoritmos heurísticos. A diferencia de la heurística, que generalmente encuentran "buenas" soluciones en un tiempo razonable, los algoritmos de aproximación proporcionan una calidad comprobable y demostrable de la solución en los límites de tiempo de corrida.

Los algoritmos heurísticos encuentran "buenas" soluciones en instancias de problema de gran tamaño. Permiten obtener un buen rendimiento a un costo razonable en una amplia gama de problemas. En general, las *heurísticas* no ofrecen una garantía de aproximación a las soluciones obtenidas. También pueden clasificarse en dos familias: *heurísticas* específicas y *meta*heurísticas. Las *heurísticas* están hechas a la medida y diseñados para resolver un problema específico y/o instancia. Las *metaheurísticas* son algoritmos de propósito general que puede aplicarse para resolver casi cualquier problema de optimización. Pueden ser vistas como metodologías generales de nivel superior que se pueden utilizar como una estrategia de orientación en el diseño de *heurísticas* subyacentes para resolver problemas específicos de optimización [14].

2.1.4.3. METAHEURISTICAS

Las soluciones computacionales óptimas son intratables para muchos problemas de optimización de gran importancia en la industria y en la ciencia. En la práctica, por lo general se está satisfecho con las "buenas" soluciones, que se obtienen para algunos casos mediante heurísticas y para otros utilizando metaheurísticas. Las *metaheurísticas* representan una familia de técnicas de optimización aproximada que obtuvieron una gran popularidad en las últimas décadas. Se encuentran entre las técnicas más prometedoras y exitosas, proporcionan "buenas" soluciones en un tiempo razonable para resolver problemas difíciles y complejos en la ciencia y la ingeniería. Esto explica el crecimiento significativo del interés en el dominio de las *metaheurísticas*. A diferencia de los algoritmos de optimización exacta, estos algoritmos no garantizan la optimización de las soluciones obtenidas [14].

El concepto de *Metaheurística* se desprende de la definición en su esencia de la palabra *Heurística*, que se entiende por la tarea de resolver inteligentemente problemas reales usando el conocimiento disponible. El término *Heurística* proviene de una palabra griega con un significado relacionado con el concepto de encontrar y se vincula a la supuesta exclamación *Eureka* de Arquímedes al descubrir su famoso principio. El término *Metaheurística* se obtiene de anteponer a *Heurística* el sufijo *Meta* que significa “más allá" o “a un nivel superior". Los conceptos actuales de lo que es una metaheurística están basados en las diferentes interpretaciones de lo que es una forma inteligente de resolver un problema. Las *Metaheurísticas* son estrategias inteligentes para diseñar o mejorar procedimientos heurísticos muy generales con un alto rendimiento [16].

En fin, se puede concebir a las *Metaheurísticas* como estrategias aplicadas a procesos de búsqueda, donde todas las situaciones intermedias en el proceso de resolución del problema se interpretan como elementos de un espacio de búsqueda, que se van modificando a medida que se aplican las distintas operaciones diseñadas para llegar a la resolución definitiva [16]

Existen diferentes tipos de *Metaheurísticas* y los más fundamentales son: *las metaheurísticas para los métodos de relajación, para los procesos constructivos, para las búsquedas por entornos, para los procedimientos evolutivos y de memoria a corto plazo*.

A continuación se definirán cada tipo de metaheurísticas:

- ***Las Metaheurísticas de Relajación*** se refieren a procedimientos de resolución de problemas que utilizan relajaciones del modelo original (es decir, modificaciones del modelo que hacen al problema más fácil de resolver), cuya solución facilita la solución del problema original.
- ***Las Metaheurísticas Constructivas*** se orientan a los procedimientos que tratan de la obtención de una solución a partir del análisis y selección paulatina de las componentes que la forman.
- ***Las Metaheurísticas de Búsqueda*** guían los procedimientos que usan transformación de sus movimientos para recorrer el espacio de soluciones alternativas y explotar las estructuras de entornos asociadas, dentro de estas se encuentran las siguientes:
    - *De Búsqueda local*: Establecen pautas de selección iterativa de las soluciones vecinas a la actual que dan lugar a búsquedas heurísticas de alto rendimiento.
    - *De Búsqueda Global:* Emplean diversos métodos para escapar de óptimos locales tres de ellos son:
        - Recomenzar la búsqueda desde otra posición del espacio de soluciones. (metaheurísticas de arranque múltiple o Multistart).
        - Modificar la estructura del entorno de una solución (Búsqueda de entornos variables o Variable neigborhodod search).
        - Permitir soluciones que no mejoran el valor de la función objetivo de la solución actual con cierta probabilidad.

- *De Búsquedas basadas en poblaciones:* En lugar de recorrer las soluciones de un único punto, es una población de soluciones iníciales la que recorre el espacio de manera simultánea e interactuando entre ellos. Contemplan mecanismo operadores para generar nuevas soluciones a partir de las ya existentes.

- ***Las Metaheurísticas Evolutivas:*** Son un conjunto de soluciones que evolucionan en el espacio de búsqueda estableciendo procedimientos de guiado del proceso. Se distinguen por la forma en que combinan la información de las diferentes soluciones para obtener otras nuevas.
- ***Las Metaheurísticas de memoria a corto plazo:*** Emplean la memoria del proceso de búsqueda para mejorar su rendimiento [16].

Las metaheurística más utilizadas en la literatura especializada son las siguientes:

- *Recocido Simulado o Simulating Annealing*
- *Algoritmos Genéticos*
- *La Búsqueda Tabú (Tabu Search)*
- *La Colonia De Hormigas (Ants Colony Optimization).*

El Recocido Simulado o Simulated Annealing, proviene de la estrecha analogía que guarda con el proceso del recocido tal y como se usa en metalurgia. Este consiste en que un metal fundido se va enfriando lentamente de manera que sus moléculas van adoptando poco a poco una configuración de mínima energía. Cuando comienza el proceso, a alta temperatura, las moléculas vibran y se desplazan caóticamente adoptando todo tipo de configuraciones en la estructura del metal de la que forman parte. A medida que la temperatura disminuye se va ralentizando el movimiento de las moléculas y estas, de acuerdo con la Termodinámica, tienden a adoptar paulatinamente las configuraciones de menor energía, siendo esta nula en el cero absoluto. El Recocido Simulado intenta realizar numéricamente un proceso análogo al del recocido metalúrgico. El espacio de configuraciones no vendría ya dado por las posiciones de las moléculas, sino por los valores de una variable de interés, que en el caso del

Flexible Flow Shop seria la secuencia de tareas que se desean procesar, mientras que el papel de la energía lo asumirla la función que se intenta minimizar, costo o tiempo de realización de tarea.

Los Algoritmos Genéticos imitan el procedimiento de la selección natural sobre el espacio de soluciones del problema considerado. Se basan en la creación de generaciones sucesivas de individuos representativos de posibles soluciones al problema. Los nuevos individuos se generan cruzando parejas seleccionada dando mayor probabilidad a aquellas soluciones que mejor valor de la función objetivo han obtenido en la generación anterior (individuos más adaptados al entorno) [17].

Por su parte, *La Búsqueda Tabú* (Tabu Search) es una técnica de optimización iterativa global, que consiste en un procedimiento determinístico que restringe la búsqueda y evita los mínimos locales, almacenando la historia de búsqueda en memoria. Se prohíben movimientos entre vecinos que cumplan ciertas propiedades, con objeto de guiar el proceso de búsqueda para no duplicar soluciones previamente obtenidas. Una función de memoria a corto plazo permite “olvidos estratégicos" convirtiendo en “prohibido" los t movimientos más recientes. Sin embargo, el estado de un movimiento no es absoluto, porque es posible seleccionar un movimiento tabú si alcanza un determinado nivel de calidad. También se dispone de funciones de memoria a largo plazo que pueden aplicarse para proporcionar una exploración más amplia del espacio de búsqueda. Finalmente existen estrategias a medio plazo o intermedias basadas en la modificación de las reglas de elección que favorecen la elección de movimientos y soluciones consideradas buenas históricamente, de forma que crean zonas de atracción del dominio de búsqueda e intensifican la búsqueda en dichas regiones. Los métodos a largo plazo diversifican la búsqueda en tareas no exploradas previamente.[17]

Por último, *La Colonia De Hormigas* se basa en la observación de hormigas reales, debido a que estos insectos tienen características de comportamiento que han llamado la atención de varios investigadores. Uno de estos aspectos es la

capacidad que tiene la colonia para encontrar la ruta más corta entre el hormiguero y las fuentes de alimento. Mientras hacen sus recorridos las hormigas depositan en la tierra una sustancia química llamada feromona, formando en el camino un rastro de dicha sustancia. Las hormigas pueden percibir la feromona y cuando tienen que escoger una ruta se deciden por aquella que tenga el rastro más fuerte. El rastro de feromonas también les permite encontrar el camino de regreso al hormiguero (ó a la fuente de alimento) y transmitir esta información a otras hormigas [18].

2.1.5. METAHEURISTICA VIRAL SYSTEMS

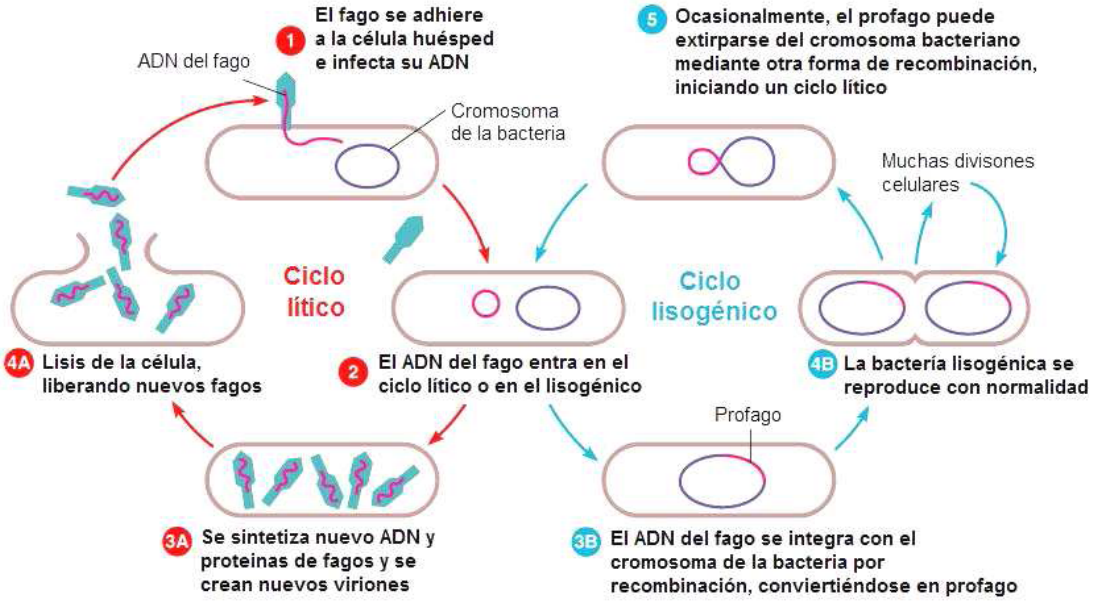
En la actualidad, en muchas situaciones se presentan problemas combinatorios y las herramientas de solución disponibles presentan debilidades, los algoritmos con alta rigurosidad matemática como Branch and Bound (B&B), resuelven el problema de forma exacta pero son demasiado lentos, y las Heurísticas superan en tiempo a los algoritmos exactos pero su solución es lejana al óptimo.

Para solucionar este inconveniente, se hace uso de las metaheurísticas algunas de estas ampliamente descritas anteriormente y de un concepto como son los Sistemas de Inteligencia artificial (AIS), que son sistemas de cálculo computacional inspirados en los principios y procesos del sistema inmune de los seres vivos vertebrados, este concepto fue implementado a mediados de los años ochenta por Farmer y otros investigadores en un trabajo sobre redes inmunes.

En su lugar, *Viral System* [19] es una metaheurísticas para solucionar problemas de optimización combinatoria, que se basa en los mecanismos de réplica y procesos de infección de los virus. Para describir las generalidades de *Viral System*, se toman como objeto de estudio los virus que infectan a las bacterias llamados *Bacteriófagos* o como se conocen comúnmente como *Fagos*, estos al momento de su replicación pueden tener un ciclo *Lítico* o un ciclo *Lisogénico*.

En el ciclo Lítico, las células huéspedes del fago son destruidas tras la replicación y encapsulación de las partículas virales, de forma que los nuevos virus quedan libres para llevar a cabo una nueva infección. En el caso Lisogénico los virus se alojan en el genoma de la célula infectándola, estos permanecen allí hasta que una fuente externa como la irradiación los active para comenzar su replicación.

Ilustración 5: Paralelo entre el ciclo lítico y el lisogénico



Viral System se compone de tres componentes, un conjunto de virus, un organismo y una interacción entre ellos.

$$VS = \{Virus, Organismo, Interaccion\}$$

El componente Virus está conformado por un conjunto de Virus, cada uno con cuatro componentes

$$Virus = \{Virus_1, Virus_2, Virus_3 \ldots\ldots Virus_i\}$$

$$Virus_i = \{estado_i, entrada_i, Salida_i, proceso_i\}$$

Donde $Estado_i$ caracteriza al virus y define la célula infectada. Normalmente es la codificación matemática de la solución en términos computacionales, que también puede llamarse genoma. Un virus concreto puede producir la infección de una célula del organismo. Además, la evolución del virus dentro de la célula puede ser definida por el número de núcleos-cápsides replicados por el proceso lítico (NR), o el número de iteraciones para el proceso lisogénico (IT). Así, $Estado_i$queda definido por la tupla genoma-NR-IT.

La $Entrada_i$ identifica la información que el virus puede obtener del organismo. Dicha información se obtiene exclusivamente de las proximidades al virus. Representa la interacción con el organismo: información del organismo al virus. En términos computacionales se corresponde con la vecindad de la célula.

El componente $Salida_i$ identifica la acción que el virus puede realizar. Representa la interacción con el organismo: virus al organismo. Se corresponde con el mecanismo de selección del tipo de réplica (lítica o lisogénica) que va a tener el virus en término computacionales.

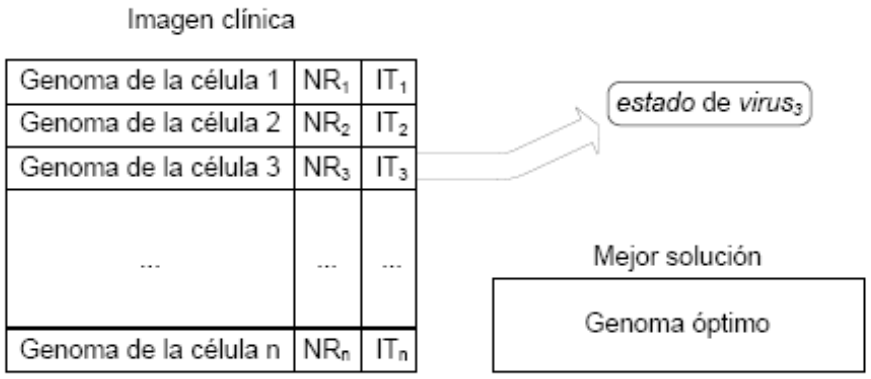
Por último, $Proceso_i$representa el comportamiento autónomo del virus, cambiando el $Estado_i$. Corresponde al proceso de replicación en términos computacionales.

El componente de *Viral System* llamado *Organismo*, está compuesto por dos componentes:

$$Organismo = \{estado_o, proceso_o\}$$

Donde el $Estado_o$caracteriza el estado del organismo en cada instante, consistente en la imagen clínica y la mejor solución. En este proyecto en realidad no se dispondrá de la mejor solución, por lo que el $Estado_o$ quedará definido exclusivamente por la imagen clínica.

Ilustración 6: Estado0, conformado por la imagen clínica y la mejor solución.



El componente $Proceso_o$ representa el comportamiento autónomo del organismo que intenta protegerse de la infección, consiste en la liberación de antígenos. Médicamente, un antígeno es una sustancia que provoca una respuesta inmune. Los antígenos generan una respuesta inmune en forma de anticuerpos que

intentan combatir la infección. Su misión computacional es liberar espacio en la población de células infectadas (imagen clínica), intentando así mantener registros libres para incorporar nuevas células infectadas. Computacionalmente, los antígenos equivalen a agentes destructivos.

El componente Interacción del *Viral System* está condicionado por las acciones de entrada y salida, que desembocan en un proceso en cada virus y a la correspondiente respuesta del Organismo. Así, el proceso de un $Virus_i$ implica un cambio en el Organismo y viceversa.

Componente de entrada. Identificación de la vecindad

El componente de entrada de cada virus,$Entrada_i$, recoge información del organismo. Se realiza un mapeo del genoma y se detecta un conjunto de células cercanas a la que se está tratando. Este conjunto se denomina vecindario de la solución factible x, $V_{(x)}$. Este vecindario depende de las restricciones del problema.

Componente de salida. Selección del tipo de réplica

El componente de salida,$Salida_i$ determina el tipo de evolución del virus. Se consideran dos tipos, ya que se trabaja con bacteriófagos: lítico si el ciclo de vida del virus se ejecuta en un único paso y lisogénico si se ejecuta en dos pasos. Para seleccionar un tipo u otro se considera que será del tipo lítico con probabilidad $prob_{liti}$ o del tipo lisogénico con probabilidad $prob_{liso}$, donde $prob_{liti} + prob_{liso} = 1$.

Proceso lítico

Cuando en una célula se repliquen un número determinado de núcleos-cápsides (LNR), es decir, cuando $NR = LNR$, el borde de la bacteria se rompe, liberando los virus almacenados. El número de núcleo-capsides replicados en cada iteración se puede aproximar por una distribución binomial dada por el máximo nivel de núcleo-capsides replicados, de LNR, y la única probabilidad de una réplica, Pr ,

$Z = Bin\,(LNR, Pr)$. El valor de LNR depende de la salud de la célula, es decir, una célula saludable (en otras palabras, con un nivel de infección bajo, lo que computacionalmente se traduce en un valor de la función objetivo bajo), tendrá bajas probabilidades de ser infectada, y por lo tanto, su valor de LNR será alto (no es conveniente infectar células que tienen valores de la infección bajos, pues costará más llegar a un óptimo a través de esta célula). Por el contrario, aquellas células con valores altos de infección tendrán bajos valores de LNR. La siguiente ecuación muestra el método de cálculo del valor del LNR en cada célula.

$$LNR_x = LNR_0\left(\frac{f(\hat{x}) - f(x)}{f(\hat{x})}\right) (Ec.No.2)$$

Donde $\hat{x}$ es la célula que produce el mejor valor conocido de la función objetivo en ese instante, x es la célula a la que se le está asignando el valor LNR y LNR_0 es un valor inicial de LNR.

Una vez que la bacteria queda abierta se comprueba la $Entrada_i$, analizando todas las opciones vecinas, eligiendo entre ellas la que dará mejor resultado en la infección del virus, es decir, la que tenga el mejor $f(v)$, y ésa es la que se infecta. Pasando a la siguiente iteración del proceso.

El organismo responde a la infección viral lanzando los antígenos. Los antígenos generan una inmune respuesta por medio de los anticuerpos. El proceso del organismo es definido por la capacidad antigénica del organismo, que depende del $State_o$ y lleva a una nueva situación del estado del organismo.

Cada una de las células infectadas en el cuadro clínico tiene una probabilidad de anticuerpos que se convierten contra la infección. La probabilidad de generar los anticuerpos para cualquier célula de la población infectada se puede dar por una distribución de probabilidad de Bernoulli.

Por lo tanto, la probabilidad de generar los anticuerpos es $p_{an}: A_{(x)} = Ber(p_{an})$, donde x es una célula infectada. Por lo tanto, la población total de células infectadas que generan anticuerpos es caracterizada por una distribución binomial

de parámetros: el tamaño de la población infectada, de n, y de la probabilidad de generar los anticuerpos, $p_{an}: A_{(population)} = Ber(n, p_{an})$

Por otra parte, sabemos que cuando un virus alcanza el límite de LNR, la frontera de la bacteria está quebrada y los nuevos virus intentan infectar nuevas células en sus vecindades. Bajo estas condiciones, debemos computar la capacidad antigénica para cada célula en las proximidades de un virus activo. Se estima como distribución de probabilidad de Bernoulli dada por la probabilidad de generar los anticuerpos, $p_{an}: A_{(x)} = Ber(p_{an}): x' \in V_{(x)}$

Por lo tanto, el número total de células en la vecindad con los anticuerpos seguirá una distribución de probabilidad binomial dada por el tamaño total de la vecindad para todos los virus activos, $|V\ (x)|$, y la probabilidad de generar los anticuerpos, $p_{an}: A = Bin(|V\ (x)|, p_{an})$.

Proceso lisogénico

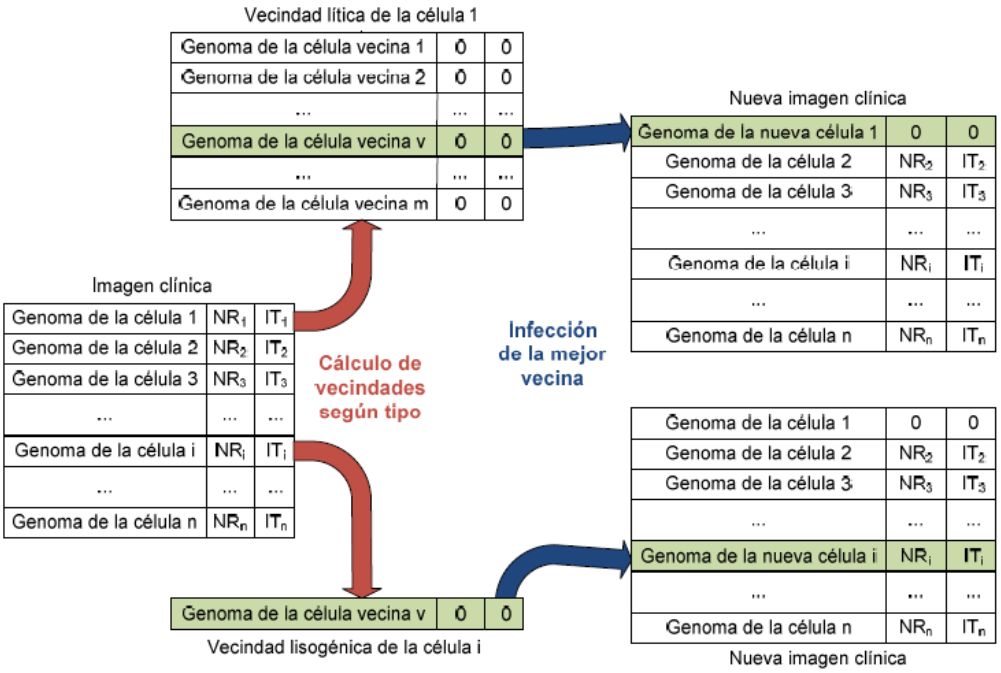
Este proceso se lleva a cabo si se ha elegido la replicación de un virus que sea del tipo lisogénico. El virus permanece en el interior de la célula infectada hasta que una causa externa le haga salir. Al igual que en el proceso lítico, esto sucede cuando han transcurrido un número determinado de iteraciones en la misma célula (LIT), es decir cuando el número de iteraciones IT sea igual a LIT. Y al igual que en el proceso lítico, el cálculo de este límite de iteraciones también se basará en cómo de saludable es la célula, por lo que el cálculo queda:

$$LIT_x = LIT_0 \left(\frac{f(\hat{x}) - f(x)}{f(\hat{x})} \right) (Ec.No.3)$$

Donde LIT_0es el valor que se le da al inicio a LIT.

El vecindario en este caso sólo consiste en un vecino, así, cuando se abre la célula después de que hayan transcurrido las debidas iteraciones, se mirará aleatoriamente un único vecino, y ese será el que se infecte, independientemente de lo bueno o malo que sea para el proceso [1].

Ilustración 7: Búsqueda de la vecindad según el tipo de replica



El fin del proceso de infección propuesto por *Viral System* termina con dos situaciones; el organismo no sobrevive a la infección es decir el organismo muere, o el organismo vence a la infección es decir logra combatirla y aislarla.

En la primera situación, se consigue cuando la diferencia entre la mejor solución encontrada y un límite más bajo conocido (LB) es más pequeña que un GAP indicado.

$$gap = \frac{|f_{(x)} - LB|}{LB} \quad (Ec.No.4)$$

La segunda situación se obtiene después de que se hayan producido un número de iteraciones, en este caso se considera que la infección no puede desarrollarse más lejos. Cuando este criterio se utiliza junto con el anterior, la situación denota que el GAP no fue alcanzado, y el virus no crea una infección seria en el organismo. Bajo esta condición, el organismo habría sobrevivido a la infección del virus.

2.2. ESTADO DEL ARTE

La investigación en la teoría de *Scheduling* ha venido desarrollándose en los últimos 40 años y ha sido objeto de mucha literatura con técnicas que van desde reglas de *"despacho"* (dispatching) no refinadas, hasta algoritmos paralelos de ramificación y poda altamente sofisticados, heurísticas basadas en cuellos de botella" (bottleneck based heuristics), y algoritmos genéticos paralelos. Además, dichas técnicas han sido formuladas desde un amplio espectro de investigadores, desde científicos de gestión, hasta expertos en producción. No obstante, con la aparición de nuevas metodologías, como redes neuronales y computación evolutiva, investigadores de campos como la bióloga, genética y neurofisiológica han tenido también contribuciones regulares a la teoría de *Scheduling*, poniendo de manifiesto la naturaleza multidisciplinar de este campo.

2.2.1.1. ALGORITMOS EXACTOS

Sin lugar a dudas los algoritmos basados en branch and bound (B&B) son los más estudiados en la literatura para solucionar de forma exacta la programación de la producción en flexible flowshop [7].

Rao [20] propuso un algoritmo B&B, para un flexible flowshop (FFS) con dos estaciones minimizando el makespan, la primera estación con una sola máquina y la segunda con dos máquinas idénticas. El caso contrario es decir dos estaciones, la primera con dos máquinas idénticas y la segunda estación con una sola maquina fue estudiado por Arthanary and Ramaswamy [21] y por Mittal and Bagga [22] ambos minimizando el makespan. Lee and Kim [23] estudiaron el problema FFS con dos estaciones y cualquier número de máquinas idénticas en paralelo en la segunda estación, propusieron un algoritmo B&B que minimiza la tardanza total. De igual forma Gupta, Hariri y Potts [24] trabajaron el problema donde la estación uno tiene n maquinas idénticas en paralelo y la estación dos solo una, en la cual

se busca minimizar el makespan. Por su parte M. Haouari, L. Hidri, A. Gharbi [25], propusieron un algoritmo B&B que resuelve el problema FFS de dos estaciones sin restricciones en el número de máquinas de cada estación, con makespan como criterio de optimización, el cual obtiene el óptimo para problemas de un tamaño de más de 1000 ordenes, sin embargo para problemas medianos entre 20 y 50 órdenes puede no encontrar el óptimo [7].

Uno de los primeros estudios para FFS genérico, es decir, con cualquier número de estaciones y cualquier número de máquinas por estación fue propuesto por Brah y Hunsucker [26], ellos desarrollaron ejemplos de un tamaño muy limitados los cuales podrían ser resueltos óptimamente, más específicamente problemas con hasta 8 órdenes de producción, dos estaciones y tres máquinas en paralelo cada una [7].

Sawik [27] propuso un modelo de programación entera mixta que encuentra la solución óptima usando CPLEX solver, para cualquier número de estaciones con cualquier número de máquinas en paralelo idénticas en cada estación, con buffer intermedios entre estaciones limitado (bloqueos), este mismo autor en [28] propone un modelo programación entera mixta parecido al anterior pero agregándole la posibilidad de tener estaciones que trabajen en batch.

Pocos trabajos se han publicado de algoritmos exactos para problemas de FFS con máquinas no relacionadas, si consideramos la formulación matemática de Jungwattanakit, Reodecha, Chaovalitwongse [6] y resolver los problemas en software como CPLEX solver, entonces se puede pensar en una forma exacta de resolver el problemas de máquinas no relacionadas, que además admite tiempos de alistamiento dependientes de las secuencia y de la máquina.

Los algoritmos exactos tienen la ventaja inherente que es la obtención del optimo, sin embargo solo se pueden aplicar a problemas muy pequeños, dado que requiere mucho tiempo de procesamiento para problemas medianos y grandes,

por lo tanto sus aplicaciones prácticas en el mundo real se ve limitada. Es aquí donde inicia las investigaciones para encontrar algoritmos eficientes que den una buena solución en tiempos satisfactorios para el usuario.

2.2.1.2. HEURÍSTICAS

Las heurísticas son algoritmos que buscan la solución de un problema mediante métodos no rigurosos, como por tanteo, reglas empíricas, etc. En el caso de la programación de la producción de FFS se pueden considerar las heurísticas más sencillas aquellas que son conocidas como reglas de programación como son Shortest Processing Time first (SPT), Earliest Due Date (EDD) first, Weighted Longest Processing Time first (LPT) y las demás reglas que se desprenden de ellas como por ejemplo WSPT en la cual cada orden tiene una importancia relativa.

Trabajos posteriores se han basado en crear nuevas heurísticas ó en mejorar las que ya han sido propuestas; uno de los primeros trabajos en FFS fue realizado por Wittrock [29] el cual estudió un problema real en la industria electrónica, propuso el modelo de programación matemática y una heurística para solucionar la programación de la producción en un sistema Flexible Flowshop con tres estaciones y máquinas en paralelo iguales en cada estación, con la particularidad que las órdenes podían saltarse el procesamiento en una o dos estaciones.

Gupta [8] además de realizar la prueba Np-hard para FFs con dos estaciones en la cual la primera estación tiene dos máquinas idénticas y la segunda estación solo una máquina, propuso un heurística muy simple para solucionarlo.

Guinet [30] trabajó una heurística ad-hoc para un caso de programación de la producción en la industrial textil, en la cual optimizaba el makespan y la tardanza promedio, el problema considerado era de m estaciones con máquinas idénticas en cada estación y tiempos de preparación dependientes de la secuencia. Este mismo autor junto con otros investigadores [31] propusieron un algoritmo

heurístico de dos pasos para optimizar el makespan en FFS de dos estaciones, esta heurística descompone el problema, primero lista de prioridades de cada una de las órdenes y luego asignan estas (siguiendo el orden de la lista) en cada una de las estaciones dependiendo de las máquinas que se encuentren disponibles. De igual forma propusieron un algoritmo [32] para FFS con cualquier número de estaciones y maquinas idénticas en cada estación, buscando optimizar el makespan o la tardanza máxima.

2.2.1.3. META HEURÍSTICAS

Durante los últimos 20 años, los investigadores en el campo de optimización combinatoria han desarrollado con éxito estrategias genéricas para mejorar el rendimiento de simples heurísticas determinísticas. Estos métodos, denominados metaheurísticas, agregan un elemento de aleatoriedad a las heurísticas determinísticas con la idea de que su uso repetido pueda llevar a mejores soluciones que las generadas de manera determinista [33].

En el caso de Flexible Flowshop, una de las primeras publicaciones fue realizada por Voss [34], donde una regla de despacho se utilizó de semilla de un algoritmo de Búsqueda Tabú para resolver un flexible flowshop de dos estaciones con una sola máquina en la segunda estación y con tiempos de preparación dependientes de la secuencia. Por su parte Haouari y M´Hallah [35], también estudiaron un Flexible Flowshop de dos estaciones pero con máquinas paralelas idénticas en cada estación. Un algoritmo de búsqueda tabú, con capacidad de almacenamiento limitado entre máquinas fue desarrollado por Wardono y Fati en [36].

Hay pocos estudios que consideran máquinas paralelas no relacionadas en cada estación. Un ejemplo de ello es Low en [37], en donde propone un método de Simulado Recocido para éste problema, y cuyo criterio es minimizar el tiempo total de flujo. Por su parte, B. Naderi et ál [38] hicieron uso de Simulado Recocido para

resolver un Flexible Flowshop con tiempos de preparación dependientes de la secuencia y con tiempo de transporte entre las estaciones de trabajo, cuyo objetivo era minimizar el tiempo total de finalización y la tardanza total.

Una heurística de búsqueda local aleatoria es propuesta por León y Ramamoorthy en [39]. Aquí, la aleatoriedad es introducida para modificar los datos del problema. Cada variante del problema original es resuelta por medio de una heurística determinística y la solución es evaluada usando los datos originales. La idea es hallar una solución a uno de los problemas modificados, que mejore la solución generada con los datos del problema original.

Los Algoritmos Genéticos, también han sido ampliamente utilizados para resolver problemas de scheduling en Flexible Flowshop. Por ejemplo, W. Xiao, et ál [40] utilizaron esta metaheurística para buscar el espacio de permutaciones para solucionar un problema de m estaciones, cuyo objetivo era minimizar el makespan. De igual forma, Ruiz y Maroto [41] utilizaron algoritmos genéticos para minimizar el makespan en un problema de m estaciones con máquinas paralelas no relacionadas, tiempos de preparación dependientes de la secuencia y elegibilidad de máquinas. Un algoritmo genético similar fue propuesto por Yaurima [42], quién además de las restricciones anteriores, también consideró capacidad de almacenamiento limitado entre las máquinas.

Otras metaheurísticas han sido menos estudiadas. Tal es el caso de Sistemas Artificiales Inmunes, la cual ha sido aplicada por M. Zandieh, et ál [43] para resolver problemas de scheduling de flexible flowshop con tiempos de preparación dependientes de la secuencia. Las Redes Neuronales fueron utilizadas por J.N.D. Gupta et ál [44] como un mecanismo para decidir entre varias heurísticas, la cual posteriormente se aplicaría en una instancia específica. Otra aplicación de redes neuronales es realizada por H. Wang et ál [45] quienes la utilizan para un problema de m estaciones con makespan como criterio de optimización. Colonia de Hormigas, ha sido también utilizada como una metaheurística para la

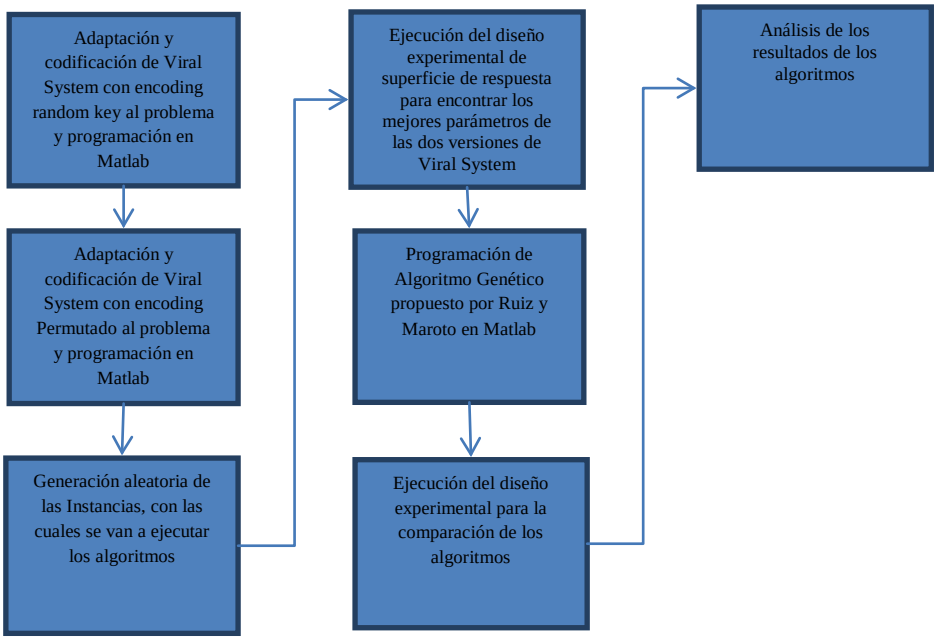
resolución de Flexible Flowshop. Ying y Lin [46] la utilizan para solucionar un problema de flexible flowshop con relaciones de precedencia. Alaykyran et ál [47], estudian un flexible flowshop estándar con makespan como criterio de optimización.

CAPÍTULO 3 METODOLOGÍA

3.1. MÉTODO Y MATERIAL

Con el objetivo de determinar la calidad de resultados que se pueden obtener al implementar un aplicativo computacional basado en *Viral System*, se hizo dos algoritmos uno utilizando un encoding ramdom key y el otro con encoding permutado. Luego se realizó un diseño experimental del *Box-Behnken*, para determinar los mejores valores que se le deben asignar a los diferentes parámetros a tener en cuenta al momento de trabajar con esta metaheurística. Para tal efecto, después de haber codificado el algoritmo se procedió a correr las réplicas requeridas con instancias generadas aleatoriamente, con base a estos resultados se hallaron los valores de los parámetros exigidos por *Viral System*, usando para tal efecto la optimización del modelo de regresión. Una vez obtenido los parámetros óptimos de Viral systems se realizó un experimento con el fin de comparar si existe diferencia estadísticamente en el makespan de los dos algoritmos Viral system y el algoritmo genético propuesto por Ruiz y Maroto [41].

Ilustración 8: Diagrama de Flujo de la Metodología



3.2. PROCEDIMIENTO E INSTRUMENTOS DE RECOLECCIÓN DE DATOS

Como se mencionó anteriormente se realizaron corridas del algoritmo aplicando el método de experimentos de *Box-Behnken*, con la ayuda de cual obtendremos el modelo de regresión que describe el comportamiento de la metaheuristica, para posteriormente optimizarla y así encontrar los mejores valores para los parámetros. Se hará uso de este método debido a que permite conseguir los parámetros óptimos dentro de toda la región de experimentación.

Para medir el desempeño de las metaheurísticas y poder compararlas entre sí, se calculó el puntaje típico estandarizado (Z). Los resultados obtenidos se tabularon para su posterior comparación y análisis (ver capítulo 4).

$$Z = \frac{valor\ alcanzado - promedio\ de\ la\ instancia}{desviacion\ estandar\ de\ la\ instancia}$$

Para medir si la diferencia significativa entre los valores arrojados por *Viral System Random Key,* Viral Systems con encoding permutado y algoritmos genéticos, se

hizo un análisis de *comparación de medianas de mood* con la ayuda de software Statgraphics, para lo cual se hará uso de las siguientes hipótesis:

$H_0: Me_1 = Me_2 = Me_3$

$H_1: Me_i \neq Me_j$ para por lo menos un par i,j

Donde

H_0: Hipótesis nula

H_1: Hipótesis alternativa

Me_1 Mediana de los resultados arrojado por *Viral System Random Key.*

Me_2 Mediana de los resultados arrojado por *Viral System Permutado*.

Me_3 Mediana de los resultados arrojado por Algoritmos Genéticos.

$Me_i \ y \ Me_j$ Son dos medianas cualesquiera de las tres enfrentadas.

CAPÍTULO 4 RESULTADOS

Para la solución del problema propuesto en este trabajo, se han creado aplicaciones computacionales para cada una de las tres metaheurísticas. Este ha sido desarrollado en el entorno de programación de *MATLAB R2009b*.

4.1. ALGORITMO VIRAL SYSTEM CON ENCODING RANDOM KEY (RK)

Es necesario definir el tipo de *encoding* utilizado, en este caso equivalente al genoma, que en términos computacionales corresponde a la solución matemática del problema, definir como se obtiene la población inicial o individuos que conforman el cuadro clínico inicial, describir también como se generan los vecinos cuando ocurren los procesos lítico y/o el lisogénico, según sea el caso, así como también el tipo de programación de las operaciones que se utilizó para generar la función de evaluación propuesta en esta investigación.

4.1.1. CODIFICACIÓN DE LAS POSIBLES SOLUCIONES (ENCODING)

Para codificar las soluciones se utilizó un encoding tipo Random-Key [48], en el que una solución es representada por un vector fila de números aleatorios; el tamaño del vector está dado por el número de ordenes (***n***) multiplicado por el número de estaciones (m), y cada elemento del vector contiene un número aleatorio en el intervalo [1, l_i+0.99], donde l_i representa el número de máquinas disponibles en la estación de trabajo i.

Por ejemplo se cuenta con tres estaciones de trabajo, la primera estación cuenta con 2 máquinas la segunda tiene una máquina y la tercera 3. Se considera que se tienen tres órdenes para ser programadas, por lo tanto una posible solución con encoding RK sería el siguiente:

Ilustración 9: Solución con encoding Random Key

Estación	1	2	3	1	2	3	1	2	3
Trabajo	1	1	1	2	2	2	3	3	3
encoding	**2,563**	**1,785**	**1,123**	**1,630**	**1,451**	**3,503**	**1,021**	**1,901**	**2,510**

En el encoding la parte entera representa la máquina donde se va realizar el trabajo y la parte decimal indica la secuencia, de esta forma para el ejemplo anterior se tiene lo siguiente:

El trabajo 1, se va a procesar en la maquina 2 de la estación 1, en la maquina 1 de la estación 2, y en la maquina 1 de estación 3. A continuación se muestra un resumen de la elegibilidad de máquina para cada orden:

Ilustración 10: Elegibilidad de la Máquina

		Estación		
		1	2	3
Trabajo	1	2	1	1
	2	1	1	3

	3	1	1	2

Para encontrar la secuencia en que serán procesadas los trabajos, se organizan los elementos del encoding de menor a mayor por estación.

Ilustración 11: Encoding Random Key con vista matricial

		Estación		
		1	2	3
Trabajo	1	2,563	1,785	1,123
	2	1,630	1,451	3,503
	3	1,021	1,901	2,510

Por lo tanto:

- ✓ En la estación 1: el trabajo 3 es el primero en ser procesado en la máquina 1 y luego se realiza el trabajo 2; mientras que en la máquina 2 se procesa la orden 1.
- ✓ En la estación 2 primero se procesa la orden 2 luego la 1 y por último la 3 (solo hay una máquina en esta estación).
- ✓ En la estación 3 se procesa una orden en cada una de las máquinas.

4.1.2. CUADRO CLÍNICO INICIAL

La población inicial o cuadro clínico inicial consiste, en un arreglo matricial de TCC filas y $n*m+4$ columnas, donde TCC es un parámetro conocido como *tamaño del cuadro clínico, n es el número de órdenes y m el número de estación*. En el arreglo matricial anterior, desde la columna 1 hasta la columna $n*m$ se representa el encoding descrito en el apartado anterior, en la columna $n*m+1$ se asigna un número aleatorio que va decidir qué tipo de réplica (lítica o lisogénica) va seguir el genoma estudiado, en la columna $n*m+2$, se ubica el número de núcleo-capsides replicados dentro de la célula o NR, en la columna $n*m+3$, se encuentra el número determinado de iteraciones en la misma célula o $Iter$, y por

ultimo en la columna $m * n + 4$, esta el valor de la función objetivo resultante del genoma estudiado. Un elemento del cuadro clínico, seria representado de la siguiente manera:

Ilustración 12: Elemento del Cuadro Clínico inicial

4.306	3.201	2.34	2.501	*Rand*	*NR*	*Iter*	$f.o(x)$

Todos los individuos que pertenecen a este cuadro clínico inicial son generados aleatoriamente.

4.1.3. REPLICACIÓN LÍTICA

En viral system la replicación lítica implica un proceso de búsqueda en vecindad, en esta investigación se utilizó el operador *Swap* en el cual se toma un elemento del genoma y se intercambia de posición con otro elemento del genoma, se realiza esta operación hasta obtener todas las posibles combinaciones. Cada intercambio de posición es un nuevo vecino.

Para realizar el operador Swap, primero se convierte el encoding en una matriz de la siguiente forma:

Se usa el encoding explicado en el capítulo 4.1.1:

encoding	**2,563**	**1,785**	**1,123**	**1,630**	**1,451**	**3,503**	**1,021**	**1,901**	**2,510**

El número de filas de la matriz es igual al número de trabajos y el número de columnas al número de estaciones:

		Estación		
		1	2	3
Trab	1	2,563	1,785	1,123
ajo	2	1,630	1,451	3,503

	3	1,021	1,901	2,510

El proceso para generar los vecinos consiste en hacer intercambio de dos filas completas y luego se convierte la matriz a un vector con la estructura del encoding.

En la siguiente figura se muestra todos los posibles vecinos de un genoma.

encoding	**2,563**	**1,785**	**1,123**	**1,630**	**1,451**	**3,503**	**1,021**	**1,901**	**2,510**

Vecinos generados:

Ilustración 13: Vecinos generados con operador Swap.

1,630	1,451	3,503	2,563	1,785	1,123	1,021	1,901	2,510
1,021	1,901	2,510	2,563	1,785	1,123	2,563	1,785	1,123
2,563	1,785	1,123	1,021	1,901	2,510	1,630	1,451	3,503

En el proceso lítico se deben generar todos los posibles vecinos, por lo cual el tamaño de la vecindad $V(x)$, viene dado por la ecuación $|V(x)| = \frac{n*(n-1)}{2}$, es decir, para un problema con $n = 10$, el tamaño de la vecindad es 45 individuos.

4.1.4. REPLICACIÓN LÍSOGENICA

La replicación lisogénica en Viral System RK*, emula el comportamiento de cambiar algunas características de los genes del virus, para esta investigación si una solución requiere una replicación lítica se genera una solución nueva con el encoding descrito en 4.1.1.*

4.1.5. CRITERIO DE PARADA DEL ALGORITMO

El algoritmo se detiene cuando pasa alguna de las condiciones siguientes:

- El número de iteraciones que ha realizado el algoritmo supera el número máximo definido por el usuario.
- Si el número de iteraciones sin mejoramiento supera el parámetro definido por el usuario.

4.1.6. PSEUDOCÓDIGO

Viral System tiene 9 parámetros de entrada, los cuales son: $\boldsymbol{TCC}$ conocidos como tamaño del cuadro clínico, $\boldsymbol{Iteraciones}$, $\boldsymbol{P_{lit}}$ define la probabilidad de réplica lítica, $\boldsymbol{P_{infec}}$, define la probabilidad de infección, $\boldsymbol{P_{replic}}$, que define la única probabilidad de réplica, $\boldsymbol{P_{anti}}$, define la probabilidad de generar antígenos, $\boldsymbol{LNR_0}$, define un valor inicial del LNR, $\boldsymbol{LIT_0}$, define un valor inicial del LIT y $\boldsymbol{It_{mejor}}$ que define el número de iteraciones sin mejoramiento.

A continuación se detalla el pseudocódigo del algoritmo propuesto:

1. Inicio del Algoritmo
2. Obtener los parámetros de *VS*: TCC, $Iteraciones$, P_{lit}, P_{infec}, P_{replic}, P_{anti}, LNR_0, LIT_0, It_{mejor}.
3. Generar el cuadro clínico inicial y asignarlo a CC, asignar en la columna $n*m+1$ de todo los genomas un número aleatorio y en las demás columnas restantes iniciar en cero.
4. Evaluar el cuadro clínico inicial ó CC y asignarle este valor en la columna $n*m+4$ de todos los genomas y guardar la mejor solución hasta este momento.

   Hacer $i=1$

   Hacer $contsinmejoramiento=0$
5. Mientras $(i <= iteraciones)\ O\ (contsinmejoramiento \leq itmejor)$

   Repetir desde $j=1$ hasta TCC

   Si $CC(j, n*m+1) < P_{\text{lit}}$

   La célula sigue proceso lítico

   Calcular LNR y NR

Si $NR \geq LNR$

Generar La vecindad $V(x)$

Calcular Y y A y haga $V(x) = Y - A$

Evaluar $V(x)$ y reemplazar en el CC el de mejor $f(x)$

Sino

Hacer $CC(j, n * m + 2) = NR$

Fin

Sino

La célula sigue proceso lisogénico

Calcular LIT

Si $LIT \geq iteraciones$

Generar la mutación, evaluarla y reemplazar en el CC

Sino

Hacer $CC(j, n * m + 3) = i$

Fin

Fin

Comparar la solución obtenida con la mejor solución hasta ahora

Si se mejoró la función objetivo

Guardar la mejor solución obtenida

Si no

$$contsinmejoramiento = contsinmejoramiento + 1$$

Fin

Fin

6. Mostrar la mejor secuencia junto con su mejor valor de $Cmax$.

7. Fin del Algoritmo.

4.1.7. DEFINICIÓN DE PARÁMETROS VIRAL SYSTEM RAMDOM KEY.

Para la determinación de la combinación óptima de parámetros del algoritmo propuesto fue necesario llevar a cabo un análisis de la superficie de respuesta para la variable $Cmax$. Se determinó hacer uso de un modelo de regresión de segundo orden, el cual exige por lo menos 3 niveles para cada factor (o

parámetro), para lo cual se hizo el diseño experimental correspondiente, escogiendo de entre los diseños de segundo orden, el de *Box-Behnken* dadas las bondades que ofrece al arrojar muy buenos resultados en pocas replicas experimentales (dado que cumple con el mínimo de 3 niveles por factor) en comparación con otros diseños.

Para garantizar la homogeneidad de las condiciones de trabajo para todas las réplicas en las instancias tenidas en cuenta para el diseño, se garantizó que *Box-Behnken* fuese por bloques completamente aleatorizados, con un bloque para cada instancia. El diseño se realizó en el software *Minitab* el cual encontró 130 diferentes condiciones experimentales para cada bloque, es decir, un total de 780 réplicas.

Tabla 3: Niveles para cada uno de los diferentes parámetros de VS

Parámetro	Niveles		
	Extremo inferior	Medio	Extremo superior
TCC	20	60	100
$iteraciones$	10	55	100
P_{lit}	0	0.5	1
P_{infec}	0	0.5	1
P_{replic}	0	0.5	1
P_{anti}	0	0.5	1
LNR_0	1	10.5	20
LIT_0	1	5.5	10
It_{mejor}	10	20	30

Luego de tener los valores de la $Cmax$ para todos los bloques en las diferentes condiciones experimentales, hubo la necesidad de hacer una estandarización de los valores, dado que se debían hacer comparaciones entre los diferentes bloques, los cuales poseen características diferentes entre sí, debido a las particularidades de cada una de las instancias.

Para analizar los datos obtenidos con el diseño experimental, se utiliza el software R Development Core Team (2010).

Como variable dependiente (variable de respuesta) en este modelo se utilizó la normalización de los datos para cada una de las instancias, con el fin de poderlas comparar en un solo análisis, la fórmula utilizada tiene siguiente forma:

$$Z_i = \frac{Cmax_i - \overline{Cmax}}{S_{Cmax}}$$

Dónde:

$Cmax_i$: El makespan obtenido en la condición experimental especifica.

$\overline{Cmax}$: Es el makespan promedio de la Instancia, recordemos que cada instancia tiene 130 condiciones experimentales.

S_{cmax}: Es la desviación estándar del makespan de cada instancia.

Cabe anotar que otra forma de poder comparar las instancias en un solo análisis de regresión, es utilizando el Gap como variable de respuesta, sin embargo en la pruebas realizadas esta presentaba un alto nivel de autocorrelacion, y los residuales no provenían de una distribución normal, por tal motivo se utilizó la variable normalizada.

Con la variable de respuesta normalizada se procede a conseguir un modelo de regresión que describa el comportamiento de la metaheuristica bajo cada uno de los parámetros. Un primer intento fue utilizando la variable Z como dependiente, resultó que no cumplía con los supuestos de normalidad y homocedasticidad, por ese motivo se procede a utilizar una transformación de la familia BOX-COX, de la que se obtiene una potencia de 3.1, de igual forma de se adicionó 6.4 a cada una de las variables Z con el fin de poder utilizar el método BoxCox, el cual exige que la variables sea positiva.

La variable de respuesta que se utilizó para el modelo de regresión es la siguiente:

$$Y = 1 + \frac{(Z + 6.4)^{3.1} - 1}{3.1}$$

Con esta variable de respuesta se obtienen los siguientes coeficientes del modelo de regresión:

Tabla 4: Resultados del Modelo de regresión para Viral System Random Key

Factor	Estimado	Error Estándar	Valor t	Pr(>|t|)
Nmax	0,254495044	0,149369679	1,703793205	0,088689885
TCC	-0,261078274	0,101265451	-2,578157414	0,010056389
Plt	196,2211283	30,84160828	6,362221013	2,87E-10
Pr	-15,55532952	4,33029285	-3,592211902	0,000341735
Pin	213,8658073	28,46914902	7,512195294	1,16E-13
Pan	20,51221741	12,62961682	1,624136164	0,104620788
LNR	2,153728801	0,718190673	2,998825913	0,002768498
LIT	4,25868072	1,476433017	2,884438826	0,003994236
ISM	5,385426502	0,642638958	8,380174328	1,53E-16
$Nmax^2$	-0,002159733	0,00128553	-1,680032699	0,093222551
Plt*Pin	-461,3381662	69,41361806	-6,64621985	4,63E-11
Pin*ISM	-10,30182769	1,518969963	-6,782114155	1,89E-11
Pan*LNR	-2,293057665	0,968282918	-2,368169079	0,018041002
Plt*Pan	67,67449603	13,74219218	4,924577908	9,69E-07
LNR*LIT	-0,195230309	0,096828292	-2,016252741	0,0440057
TCC*LIT	-0,026719052	0,017116985	-1,560967124	0,11880664
plt2*ISM	-10,33230978	1,667034954	-6,198016278	7,95E-10
plt*pin*ISM	19,11302962	3,60268875	5,305212564	1,35E-07

Error Residual estándar: 40.8 en 1146 grados de libertad.

R^2: 0.8846, R^2 ajustado: 0.8824

Estadístico F: 399.2 en 22 y 1146 grados de libertad.

p-value del modelo: < 2.2e-16

Por lo tanto el modelo de regresión está definido de la siguiente forma:

Ecuación 2: Modelo de Regresión Viral System Random Key

$$1+\frac{(Z+6.4)^{3.1}-1}{3.1} = 0{,}25Nmax - 0.26TCC + 196.22 * Plt - 15.55 * Pr + 213.86 * Pin + 20{,}51 * Pan + 2{,}15 * LNR + 4{,}25 * LIT + 5{,}38 * ISM - 0{,}002 * Nmax^2 - 461{,}34 * plt * pin - 10{,}3 * pin * ISM - 2{,}29 * Pan * LNR + 67{,}67 * plt * pan - 0{,}19 * LNR * LIT - 0{,}02 * TCC * LIT - 10{,}33 * Plt * ISM + 19{,}11 * Plt * Pin * ISM$$

Podemos observar que este modelo explica alrededor del 88% de la variabilidad del proceso, y además es significativo con una confianza del 95%.

La tabla ANOVA asociada a este modelo es la siguiente:

Tabla 5: Análisis de Varianza para modelo de regresión Viral System con Random Key

Factor	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Nmax	1	11789471,6	11789471,6	6985,81848	0
TCC	1	903677,408	903677,408	535,471523	1,4326E-97
Plt	1	180255,269	180255,269	1,07E+02	5,2982E-24
Pr	1	121077,895	121077,895	71,7443682	7,3928E-17
Pin	1	24089,6367	24089,6367	1,43E+01	0,00016611
Pan	1	477107,603	477107,603	282,708777	6,8052E-57
LNR	1	114617,401	114617,401	67,9162208	4,6068E-16
LIT	1	128051,66	128051,66	75,8766533	1,0339E-17
ISM	1	192176,535	192176,535	1,14E+02	2,0514E-25
$Nmax^2$	1	109914,055	109914,055	65,129266	1,7532E-15
Plt*Pin	1	223499,487	223499,487	1,32E+02	4,3896E-29
Pin*ISM	1	142057,688	142057,688	8,42E+01	2,0372E-19
Pan*LNR	1	41794,397	41794,397	24,7651532	7,4633E-07
Plt*Pan	1	16918,7068	16918,7068	1,00E+01	0,00158477
LNR*LIT	1	32180,36	32180,36	19,0683824	1,3748E-05
TCC*LIT	1	22441,1899	22441,1899	13,2974643	0,00027765
plt2*ISM	1	19419,2694	19419,2694	1,15E+01	0,00071699
plt*pin*ISM	1	47498,7985	47498,7985	28,1452804	1,3492E-07
Residuals	1150	1940773,64	1687,62926		

Del anterior análisis de varianza podemos observar que todos los factores y las interacciones seleccionadas son significativos con una confianza del 95%

El sofware *R Development Core Team (2010)* arrojó los P-valores y graficas (ver ANEXO III) de las pruebas de normalidad, homocedasticidad e independencia para poder validar cada uno de los supuestos. Los resultados de dichas pruebas se muestran en la Tabla No. 6.

Tabla 6: P-valores, para las diferentes pruebas de validación de supuestos

Prueba	P- valor
Normalidad: test Lilliefors (Kolmogorov-Smirnov)	0.1693
Homocedasticidad: test de Barlett	0.05135
Independencia: test de Durbin-watson	0.9575

Dado que se tomó un nivel de significancia $\alpha = 0{,}05$ y una confianza del 95% para las tres pruebas, se aceptaron los supuestos de normalidad, homocedasticidad e independencia de los factores y la variable de respuesta.

OPTIMIZACIÓN DE PARÁMETROS

Con el modelo de regresión obtenido en la sección anterior se procede a optimizar parámetros.

La variable de respuesta a pesar de ser una transformación de potencia de la variable Z, es congruente pensar que minimizando esta se minimiza de igual forma la variable transformada, por tanto el objetivo es:

Ecuación 3: Optimización de Parámetros Viral System Random Key

Minimizar:

$$1+\frac{(Z+6.4)^{3.1}-1}{3.1} = 0{,}25Nmax - 0.26TCC + 196.22 * Plt - 15.55 * Pr + 213.86 * Pin + 20{,}51 * Pan + 2{,}15 * LNR + 4{,}25 * LIT + 5{,}38 * ISM - 0{,}002 * Nmax^2 - 461{,}34 * plt * pin - 10{,}3 * pin * ISM - 2{,}29 * Pan * LNR + 67{,}67 * plt * pan - 0{,}19 * LNR * LIT - 0{,}02 * TCC * LIT - 10{,}33 * Plt * ISM + 19{,}11 * Plt * Pin * ISM$$

Sujeto a:

Nmax>= 10

Nmax<=100

TCC>= 20

TCC<=100

Plt>= 0

Plt<=1

Pr>= 0

Pr<=1

Pin>= 0

Pin<=1

Pan>= 0

Pan<=1

LNR >=1

LNR<=20

LIT>=1

LIT<=10

ISM>= 10

ISM<=30

Nmax, TCC, LNR, LIT ISM $\in Z^+$

Ejecutando este modelo en software GAMS se tienen los siguientes parámetros óptimos.

Tabla 7: Parametros Optimos de Viral Systems con Random Key

Factor	Optimo
Nmax	10
TCC	100
plt	1
pr	1
pin	1
pan	0
LNR	1
LIT	1
Tol	10

4.2. ALGORITMO VIRAL SYSTEM CON ENCODING PERMUTADO

Para la solución del problema propuesto en este trabajo, se han creado aplicaciones computacionales para cada una de las tres meta heurísticas. Este ha sido desarrollado en el entorno de programación de *MATLAB R2009b*.

Es necesario definir el tipo de *encoding* utilizado, en este caso equivalente al genoma, que en términos computacionales corresponde a la solución matemática del problema, definir como se obtiene la población inicial o individuos que conforman el cuadro clínico inicial, describir también como se generan los vecinos cuando ocurren los procesos lítico y/o el lisogénico, según sea el caso, así como también el tipo de programación de las operaciones que se utilizó para generar la función de evaluación propuesta en esta investigación.

4.2.1. CODIFICACIÓN DE LAS POSIBLES SOLUCIONES (ENCODING)

El genoma propuesto en este algoritmo, consiste en un vector fila cuyo número de columnas es igual al número de órdenes n del problema, donde cada casilla de este vector representa un trabajo a realizar, el genoma es definido finalmente por una permutación de los trabajos. La figura 9 ilustra lo anteriormente descrito.

Suponga que se tiene un problema con $n = 5$, el genoma propuesto es representado de la siguiente manera:

Ilustración 14: Ilustración del genoma propuesto

3	5	2	1	4

4.2.2. CUADRO CLÍNICO INICIAL

La población inicial o cuadro clínico inicial consiste, en un arreglo matricial de TCC filas y $n + 4$ columnas, donde TCC es un parámetro conocido como *tamaño del cuadro clínico*. En el arreglo matricial anterior, desde la columna 1 hasta la columna n se representa el genoma permutado, en la columna $n + 1$ se asigna un número aleatorio que va decidir qué tipo de réplica (lítica o lisogénica) va seguir el genoma estudiado, en la columna $n + 2$, se ubica el número de núcleo-capsides replicados dentro de la célula o NR, en la columna $n + 3$, se encuentra el número determinado de iteraciones en la misma célula o $Iter$, y por ultimo en la columna $n + 4$, esta el valor de la función objetivo resultante del genoma estudiado. Un elemento del cuadro clínico con $n = 4$, seria representado de la siguiente manera:

Ilustración 15: Elemento del Cuadro Clínico inicial

4	3	1	2	*Rand*	*NR*	*Iter*	$f.o(x)$

Todos los individuos que pertenecen a este cuadro clínico inicial son generados aleatoriamente, es decir, para cada fila del cuadro clínico se agrega un genoma permutado obtenido de manera aleatoria y posteriormente se le agregan las demás columnas anteriormente descritas.

Una solución es generada por la heurística de NEH creada por Nawaz, Enscore y Ham[49]. Modificada para flexible flowshop. El algoritmo es como sigue:

Paso 1. Se genera una solución aleatoria inicial.

Paso 2. Tomar los dos primeros trabajos y programarlos de manera que se minimice makespan parcial.

Paso 3. Para i=3 hasta n hacer

Paso 4. Insertar el i-ésimo trabajo en alguna posición de las i posiciones disponibles, de manera que el lugar en que se quede fijo minimice la duración del makespan parcial.

4.2.3. REPLICACIÓN LÍTICA

En viral system la replicación lítica implica un proceso de búsqueda en vecindad, en esta investigación se utilizó el operador *Swap* en el cual se toma un elemento del genoma y se intercambia de posición con otro elemento del genoma, se realiza esta operación hasta obtener todas las posibles combinaciones. Cada intercambio de posición es un nuevo vecino. En la siguiente figura se muestra todos los posibles vecinos de un genoma.

4	3	1	2	Genoma Original

Vecinos generados:

Ilustración 16: Vecinos generados con operador Swap.

3	4	1	2
1	3	4	2
2	3	1	4

4	1	3	2
4	2	1	3
4	3	2	1

Figura No. 1 Vecinos generados con operador *Swap*.

En el proceso lítico se deben generar todos los posibles vecinos, por lo cual el tamaño de la vecindad $V(x)$, viene dado por la ecuación $|V(x)| = \frac{n*(n-1)}{2}$, es decir, para un problema con $n = 10$, el tamaño de la vecindad es 45 individuos.

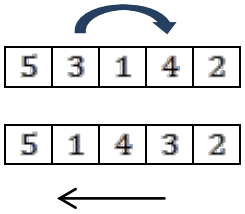
4.2.4. REPLICACIÓN LÍSOGENICA

La replicación lisogénica en Viral System es realmente un operador de mutación, en esta investigación se utilizó el operador *Shift,* en el cual se generan dos

números enteros aleatorios a y b, entre 1 y el número de órdenes n, estos indican las posiciones que van a cambiar, si $a < b$, el elemento que se encuentra en la posición a pasa a la posición b, desplazando hacia la izquierda todos los elementos; en el caso en que no se cumpla la condición $a < b$, el elemento que se encuentra en la posición a pasa a la posición b, desplazando hacia la derecha todos los elementos. Suponga que tiene un genoma de $n = 5$:

$a = 2, b = 4$, en este caso a es menor que b, en la Figura No. 12 se ve gráficamente la mutación descrita:

Ilustración 17: Mutación Shift para cuando a<b



Ahora, si $a = 4, b = 2$, en este caso a es mayor que b, en la Figura No. 13 se ve gráficamente la mutación descrita:

Ilustración 18: Mutación Shift para cuando a>b

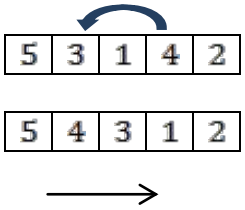


Figura No. 2 Mutación *Shift* para cuando $a > b$

En el proceso lisogénico se escoge un solo vecino teniendo en cuenta la operación anteriormente descrita.

4.2.5. ELEGIBILIDAD DE MAQUINA

La forma como se programan las operaciones que se proponen en esta investigación, se basó en el algoritmo propuesto por Ruiz y Maroto (2006), en donde la orden es procesada en la máquina que le representa el menor tiempo de finalización parcial, en este se considera los tiempos de procesamiento, los tiempos de preparación, y el tiempo de liberación. El tiempo de finalización parcial es calculado de la siguiente forma:

$$C_{i,\pi(j)} = \min_{l \in E_{ij}} \{ \max \left\{ C_{i,Li_l} + S_{il,L_{il},\pi(j)}; C_{i-1,\pi(j)}; r_{\pi_{(j)}} \right\} + p_{il,\pi_{(j)}} \}$$

Dónde:

i: denota la estación.

l: denota la máquina.

$\pi_{(j)}$: hace alusión al trabajo que se encuentra en la posición j de la secuencia

L_{il}: La última orden programada en la maquina l de estación i.

$C_{i,\pi(j)}$: Tiempo de finalización en la estación i, del trabajo que se encuentra en la posición j de la secuencia.

C_{i,Li_l}: Tiempo de finalización del último trabajo programado en la maquina l de la estación i.

$S_{il,L_{il},\pi(j)}$: Tiempo de alistamiento de la orden $\pi_{(j)}$ en la maquina l de la estación i.
$C_{i-1,\pi(j)}$: Tiempo de finalización en la estación anterior, del trabajo que se encuentra en la posición j de la secuencia.

$r_{\pi_{(j)}}$: Tiempo de liberación de la orden $\pi_{(j)}$.
$p_{il,\pi_{(j)}}$: Tiempo de procesamiento de la orden $\pi_{(j)}$ en la maquina l de la estación i.

4.2.6. CRITERIO DE PARADA DEL ALGORITMO

El algoritmo se detiene cuando pasa alguna de las condiciones siguientes:

- El número de iteraciones que ha realizado el algoritmo supera el número máximo definido por el usuario.

- Si el número de iteraciones sin mejoramiento supera el parámetro definido por el usuario.

4.2.7. PSEUDOCÓDIGO

Viral System tiene 9 parámetros de entrada, los cuales son: $\boldsymbol{TCC}$ conocido como tamaño del cuadro clínico, $\boldsymbol{Iteraciones}$, $\boldsymbol{P_{lit}}$ define la probabilidad de réplica lítica, $\boldsymbol{P_{infec}}$, define la probabilidad de infección, $\boldsymbol{P_{replic}}$, que define la única probabilidad de réplica, $\boldsymbol{P_{anti}}$, define la probabilidad de generar antígenos, $\boldsymbol{LNR_0}$, define un valor inicial del LNR, $\boldsymbol{LIT_0}$, define un valor inicial del LIT y $\boldsymbol{It_{mejor}}$ que define el numero de iteraciones sin mejoramiento.

A continuación se detalla el pseudocódigo del algoritmo propuesto:

8. Inicio del Algoritmo
9. Obtener los parámetros de *VS*: TCC, $Iteraciones$, P_{lit}, P_{infec}, P_{replic}, P_{anti}, LNR_0, LIT_0, It_{mejor}.
10. Generar el cuadro clínico inicial y asignarlo a CC, asignar en la columna $n+1$ de todo los genomas un número aleatorio y en las demás columnas restantes iniciar en cero.
11. Evaluar el cuadro clínico inicial ó CC y asignarle este valor en la columna $n+4$ de todos los genomas y guardar la mejor solución hasta este momento.

    Hacer $i = 1$
12. Mientras $i <= \ iteraciones$

    Repetir desde $j = 1$ hasta TCC

    Si $CC(j, n+1) < P_{\text{lit}}$

    La célula sigue proceso lítico

    Calcular LNR y NR

    Si $NR \geq LNR$

    Generar La vecindad $V(x)$

    Calcular Y y A y haga $V(x) = Y - A$

    Evaluar $V(x)$ y reemplazar en el CC el de mejor $f(x)$

    Sino

Hacer $CC(j, n+2) = NR$

Fin

Sino

La célula sigue proceso lisogénico

Calcular LIT

Si $LIT \geq iteraciones$

Generar la mutación, evaluarla y reemplazar en el CC

Sino

Hacer $CC(j, n+3) = i$

Fin

Fin

Comparar la solución obtenida con la mejor solución hasta ahora

Guardar la mejor solución obtenida

Fin

13. Mostrar la mejor secuencia junto con su mejor valor de $Cmax$.

14. Fin del Algoritmo.

4.2.8. DEFINICIÓN DE PARÁMETROS VIRAL SYSTEM ENCODING PERMUTADO.

La determinación de la combinación óptima de parámetros del algoritmo propuesto se desarrolló bajo la misma metodología descrita en la sección 4.1.8.

La variable de respuesta que se utilizó para el modelo de regresión es la siguiente:

$$Y = 1 + \frac{(Z + 4.5)^{2.2} - 1}{2.2}$$

Con esta variable de respuesta se obtienen los siguientes coeficientes del modelo de regresión:

Tabla 8: Coeficientes del Modelo de Regresion Viral System con encoding permutado

Factor	Estimate	Std. Error	t value	Pr(>|t|)

Nmax	-0,04454229	0,01881716	-2,36711056	0,01817821
TCC	0,35383698	0,04236223	8,35265133	3,17E-16
plt2	42,9692934	6,09688021	7,04775096	4,10E-12
pr	-8,10832483	2,16708387	-3,7415833	0,00019669
pin2	-0,82363744	2,89629628	-0,2843761	0,77619994
pan	-7,711713	3,26445844	-2,36232537	0,01841279
LNR	0,06368433	0,13310079	0,47846692	0,63245607
Tol	1,01987129	0,12649763	8,06237462	2,92E-15
I(LNR^2)	-0,01354031	0,00434684	-3,11497926	0,00190896
I(pr^2)	5,31842132	1,73873498	3,05878778	0,00230069
I(plt2^2)	2,77507424	1,73873498	1,5960306	0,11089931
I(pan^2)	-2,89347319	1,73873498	-1,66412548	0,0965012
I(pin2^2)	-2,58222727	1,76928375	-1,45947606	0,14484896
TCC:Tol	-0,01707349	0,00236786	-7,2105257	1,35E-12
plt2:pin2	-10,2030398	3,09197411	-3,29984644	0,00101265
TCC:plt2	-0,77325108	0,11598356	-6,66690273	5,04E-11
plt2:pan	8,46606688	2,18635586	3,87222732	0,00011716
plt2:Tol	-2,27468227	0,35009065	-6,49740936	1,48E-10
Nmax:pan	0,09568483	0,03435527	2,78515753	0,00548387
pin2:pan	8,34994069	3,09197411	2,7005209	0,00707779
pr:LNR	0,21468393	0,10931779	1,96385165	0,04991288
plt2:LNR	0,29671715	0,15459871	1,91927321	0,05532542
TCC:plt2:Tol	0,03550712	0,00611269	5,80875742	9,27E-09

Error Residual estándar:: 5.355 en 757 grados de libertad

R^2: 0.8753, R^2 ajustado: 0.8716

Estadístico F: 231.1 en 23 and 757 grados de libertad,

p-value del modelo: < 2.2e-16

Por lo tanto el modelo de regresión está definido de la siguiente forma:

Ecuación 4: Modelo de Regresión Viral System Permutado

$$1+\frac{(Z+4.5)^{2.2}-1}{2.2} = -0.04454229 * Nmax + 0.35383698 * TCC + 42.9692934 * plt - 8.10832483 * pr - 0.82363744 * pin - 7.711713 * pan + 0.06368433 * LNR + 1.01987129 * \text{ISM} - 0.01354031 * LNR^2 + 5.31842132 * pr^2 + 2.77507424 * plt^2 - 2.89347319 * pan^2 - 2.58222727 * pin^2 - 0.01707349 * TCC * \text{ISM} - 10.2030398 * plt * pin - 0.77325108 * TCC * plt + 8.46606688 * plt * pan - 2.27468227 * plt * \text{ISM} + 0.09568483 * Nmax * pan + 8.34994069 * pin * pan + 0.21468393 * pr * LNR + 0.29671715 * plt * LNR + 0.03550712 * TCC * plt * \text{ISM}$$

Se puede observar que este modelo explica alrededor del 87% de la variabilidad del proceso, y además es significativo con una confianza del 95%.

La tabla ANOVA asociada a este modelo es la siguiente:

Tabla 9: Análisis de Varianza Viral system con encoding permutado

Factor	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Nmax	1	123410,642	123410,642	4302,88423	1,651e-314
TCC	1	11881,5554	11881,5554	414,267007	8,68E-74
plt2	1	1642,34935	1642,34935	57,2628016	1,11E-13
pr	1	1503,82253	1503,82253	52,4328707	1,10E-12
pin2	1	40,3023899	40,3023899	1,40519906	0,236226991
pan	1	3496,48754	3496,48754	121,909916	2,20E-26
LNR	1	1361,16183	1361,16183	47,4588064	1,18E-11
Tol	1	1262,36367	1262,36367	44,0140705	6,21E-11
I(LNR^2)	1	916,440509	916,440509	31,952977	2,24E-08
I(pr^2)	1	19,0906351	19,0906351	0,66562163	0,414838918
I(plt2^2)	1	113,977214	113,977214	3,97397459	0,046567027
I(pan^2)	1	615,911341	615,911341	21,4746082	4,22E-06
I(pin2^2)	1	1001,71125	1001,71125	34,9260602	5,18E-09
TCC:Tol	1	2486,33535	2486,33535	86,6895509	1,35E-19
plt2:pin2	1	692,672253	692,672253	24,1509845	1,09E-06
TCC:plt2	1	352,589452	352,589452	12,2935232	0,000481307
plt2:pan	1	268,935734	268,935734	9,3768196	0,002275245
plt2:Tol	1	235,363585	235,363585	8,20627977	0,004290156
Nmax:pan	1	66,8728793	66,8728793	2,33161624	0,127187887
pin2:pan	1	72,0029887	72,0029887	2,51048466	0,113508786
pr:LNR	1	36,8999284	36,8999284	1,28656749	0,257040536
plt2:LNR	1	14,4979119	14,4979119	0,50548993	0,477316777

TCC:plt2:Tol	1	967,741647	967,741647	33,7416627	9,27E-09
Residuals	757	21711,4501	28,6809116		

Del anterior análisis de varianza se puede observar que todos los factores y las interacciones seleccionadas son significativos con una confianza del 95%

El sofware *R Development Core Team (2010)* arrojó los P-valores y graficas (ver ANEXO II) de las pruebas de normalidad, homocedasticidad e independencia para poder validar cada uno de los supuestos. Los resultados de dichas pruebas se muestran en la Tabla No. 9.

Tabla 10: Pruebas de validacion de supuestos Viral systems con encoding permutado

Prueba	P- valor
Normalidad: test Lilliefors (Kolmogorov-Smirnov)	0.06804
Homocedasticidad: test de Barlett	0.05177
Independencia: test de Durbin-watson	0.20

Dado que se tomó un nivel de significancia $\alpha = 0{,}05$ y una confianza del 95% para las tres pruebas, se aceptaron los supuestos de normalidad, homocedasticidad e independencia de los factores y la variable de respuesta.

OPTIMIZACIÓN DE PARÁMETROS

Con el modelo de regresión obtenido en la sección anterior se procede a optimizar parámetros.

La variable de respuesta a pesar de ser una transformación de potencia de la variable Z, es congruente pensar que minimizando esta se minimiza de igual forma la variable transformada, por tanto el objetivo es:

Minimizar:

Ecuación 5: Optimización de Parámetros Viral System Permutado

$$1+\frac{(Z+4.5)^{2.2}-1}{2.2} = -0.04454229 * Nmax + 0.35383698 * TCC + 42.9692934 * plt - 8.10832483 * pr - 0.82363744 * pin - 7.711713 * pan + 0.06368433 * LNR + 1.01987129 * \text{ISM} - 0.01354031 * LNR^2 + 5.31842132 * pr^2 + 2.77507424 * plt^2 - 2.89347319 * pan^2 - 2.58222727 * pin^2 - 0.01707349 * TCC * \text{ISM} - 10.2030398 * plt * pin - 0.77325108 * TCC * plt + 8.46606688 * plt * pan - 2.27468227 * plt * \text{ISM} + 0.09568483 * Nmax * pan + 8.34994069 * pin * pan + 0.21468393 * pr * LNR + 0.29671715 * plt * LNR + 0.03550712 * TCC * plt * \text{ISM}$$

Sujeto a:

Nmax>= 10

Nmax<=100

TCC>= 20

TCC<=100

Plt>= 0

Plt<=1

Pr>= 0

Pr<=1

Pin>= 0

Pin<=1

Pan>= 0

Pan<=1

LNR >=1

LNR<=20

LIT>=1

LIT<=10

ISM>= 10

ISM<=30

Nmax, TCC, LNR, LIT ISM $\in Z^+$

Ejecutando este modelo en software GAMS se tienen los siguientes parámetros óptimos.

Tabla 11: Parámetros óptimos de Viral System con encoding permutado

Factor	Optimo
Nmax	100
TCC	100
plt	1
pr	0.742
pin	1
pan	0
LNR	1
LIT	1
Tol	10

4.3. ALGORITMO GENÉTICO

El algoritmo genético utilizado, se basó en el algoritmo propuesto por Ruiz y Maroto (2006), este ha sido desarrollado en el entorno de programación de *MATLAB R2009b*.

4.3.1. CODIFICACIÓN DE LAS POSIBLES SOLUCIONES (ENCODING)

El genoma propuesto en este algoritmo, consiste en un vector fila cuyo número de columnas es igual al número de órdenes n del problema, donde cada casilla de este vector representa un trabajo a realizar, el genoma es definido finalmente por una permutación de los trabajos. La figura 9 ilustra lo anteriormente descrito. Suponga que se tiene un problema con $n = 5$, el genoma propuesto es representado de la siguiente manera:

Ilustración 19: Ilustración del genoma propuesto para Algoritmo Genetico

3	5	2	1	4

4.3.2. POBLACIÓN INICIAL

La población inicial consiste, en un arreglo matricial de $TPob$ filas y $n+1$ columnas, donde $TPob$ es un parámetro conocido como *tamaño de la población*. En el arreglo matricial anterior, desde la columna 1 hasta la columna n se representa el genoma permutado, en la columna $n+1$, esta el valor de la función objetivo resultante del genoma estudiado. Una solución de la población con $n=4$, seria representado de la siguiente manera:

Ilustración 20: Elemento de la población inicial

4	3	1	2	$f.o(x)$

Todos los individuos que pertenecen a esta población inicial son generados aleatoriamente. Menos una solución la cual es generada con la heurística NEH modificada para Flexible Flowshop ver sección 4.2.2.

4.3.3. SELECCIÓN

La selección de los padres que serán posteriormente cruzados, se realiza mediante Torneo, en el cual se eligen aleatoriamente cuatro candidatos de la población inicial, los dos primeros compiten y se escoge como primer padre aquel que tenga el makespan más bajo, de igual forma se realiza con el segundo padre el cual es el ganador entre los candidatos tres y el cuatro.

4.3.4. CRUZAMIENTO

El operador de cruzamiento seleccionado fue SB2OX “Similar Block 2-Point Order Crossover’’ [50], ya que este fue el operador con mejores resultados en el trabajo realizado por Ruiz [41].

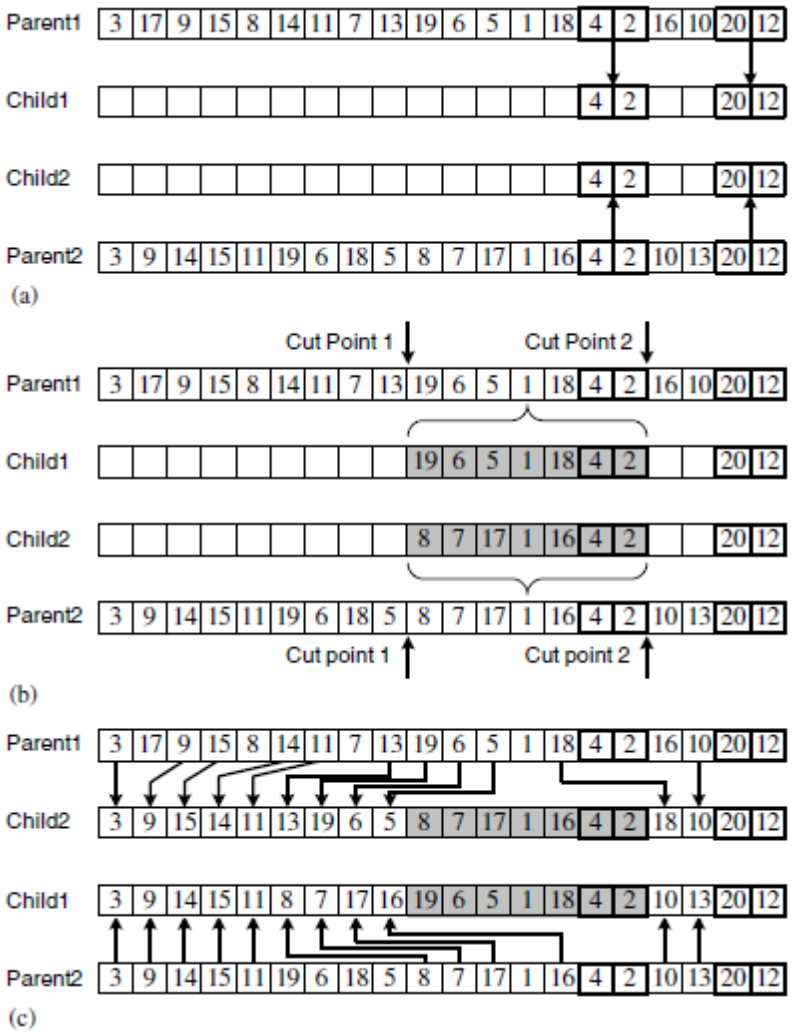
El SB2OX consiste en lo siguiente:

Primero ambos padres son examinados posición a posición básica, si se encuentran bloques de al menos dos trabajos consecutivos idénticos, todo el

bloque se copia directamente a los hijos en la misma posición donde se encontraba en los padres. (Ilustración 21a).

Luego se generan dos puntos de corte aleatorios, la sección de entre esos dos puntos se copia directamente al hijo es decir la sección del padre uno se copia al hijo uno y se hace lo mismo con el padre y el hijo dos (Ilustración 21b). Por último los elementos faltantes en los hijos son copiados en orden relativo al otro padre, esto quiere decir que los elementos faltantes del hijo uno se diligencia del padre dos y se hace igual con el hijo dos y el padre uno (Ilustración 21c).

Ilustración 21: Cruzamiento SB2OX



4.3.5. MUTACIÓN

En esta metaheurística se utilizó el operador *Shift, la cual fue descrita en la sección 4.2.4.*

4.3.6. ELEGIBILIDAD DE MAQUINA

La forma como se programan las operaciones que se proponen en esta investigación, se basó en el algoritmo propuesto por Ruiz y Maroto (2006), la cual fue descrita en la sección 4.2.5.

4.3.7. RECOMBINACIÓN

La recombinación se utiliza para superar los problemas ocasionados por los algoritmos genéticos, cuando la población alcanza una baja diversidad y las soluciones generadas tienden a un óptimo local:

La recombinación utilizada en este trabajo se ejecuta si el mejor makespan encontrado permanece sin cambios durante Gr generaciones, en ese caso se realiza el siguiente procedimiento:

Se ordena la población en orden creciente al Cmax.

- El 20% de los individuos de la secuencia ordenada se mantienen.
- El 40% son reemplazados por mutación SHIFT del primer 20% de la lista ordenada.
- El 20% son reemplazados por una nueva secuencia del NEHT RMB (ver 4.3.2).
- 20% de los individuos restantes se generan aleatoriamente.

4.3.8. CRITERIO DE PARADA DEL ALGORITMO

El algoritmo se detiene cuando pasa alguna de las condiciones siguientes:

- El número de iteraciones que ha realizado el algoritmo supera el número máximo definido por el usuario.

- Si el número de iteraciones sin mejoramiento supera el parámetro definido por el usuario.

4.3.9. PSEUDOCÓDIGO

El algoritmo genético utilizado tiene 5 parámetros de entrada, los cuales son: $Tpob$ conocidos como tamaño de la población, $Iteraciones$, P_c define la probabilidad de cruzamiento, P_m, define la probabilidad de mutación, G_r, que define el número de iteraciones sin mejoramiento para iniciar la recombinación.

A continuación se detalla el pseudocódigo del algoritmo propuesto:

```
Inicio del Algoritmo
    Obtener los parámetros del algoritmo:  Tpob, Iteraciones, Pc, Pm, Gr.
    Pob=generación_Poblacion(Tpob)
    Pob= evaluacion(Pob);
    Hacer i = 1
   Mientras no se cumpla el criterio de parada
mientras popcount<Tpob
        Padres=seleccion(Pob,2); % Selecciona dos padres  %Cruzamiento SB2OX
        genera dos hijos
        Si aleatorio()<=pc
            Hijos=Cruzamiento(Padres(1),Padres(2));
        Sino
            Hijos=Padres;
        Fin si
        %Mutacion SHIFT
        Para i=1 hasta 2
            Si aleatorio()<=pm
                Hijos(i)=mutacion(Hijos(i));
            Fin si
        Fin para
        %Evaluacion de los Hijos
        Hijos=evaluacion(Hijos);
        %Agregar los hijos a la población si son mejores que el peor
        Para i=1 hasta 2
            Si PeorCmax(Pob)>=(Hijos(i,no+1))
                Pob(posición_del_Peor,:)=Hijos(i,:);
            Fin si
        Fin para
        popcount=popcount+2;
 Fin mientras
```

```
        Si mejorencontrado > mejor(pob)

                mejor_encontrado=mejor(pob)
                contador=0;
        Sino
                contador=contador+1
        Fin si

        Si contador >= Gr
                Pob=recombinar(Pob)
                Contador=0
        Fin si
    Fin mientras
  Mostrar la mejor secuencia junto con su mejor valor de Cmax.

Fin del Algoritmo.
```

4.3.10. PARAMETRIZACIÓN DEL ALGORITMO GENÉTICO

Los parámetros escogidos de este algoritmo se tomaron del articulo presentado por [41], el cual son: 100 Iteraciones, Probabilidad de mutación es de 0.01, Probabilidad de cruzamiento es igual a 0.1 y por último número de iteraciones sin mejoramiento antes de recombinar es 50.

4.4. DISEÑO EXPERIMENTAL DE COMPARACIÓN DE ALGORITMOS

Las instancias fueron generadas aleatoriamente con la siguiente configuración: se generaron instancias para las 20, 50, 100 órdenes de producción; el número de estaciones de trabajo se generaron con 5, 10, 20 y teniendo 1, 2, 3 máquinas por estación, de igual forma se generó un cuarto tipo en la cual se genera aleatoriamente el número de máquinas en cada una de las estaciones con una distribución uniforme entre 1 y 3. De esta forma se crearon 36 instancias con todas las combinaciones.

Los tiempos de procesamiento se generaron con una distribución uniforme con mínimo 1 y máximo 99, de igual forma los tiempos de alistamiento se generaron con mínimo 0 y máximo 99.

Después de determinar los mejores niveles de los parámetros para cada uno de los algoritmos, se precedió a efectuar las comparaciones del desempeño de *Viral System* RK contra un algoritmo Viral System Permutado y *Algoritmo Genético*.

El problema fue evaluado en 36 instancias aleatorias un total de 5 veces por cada algoritmo (Ver *ANEXO 3: Determinación del tamaño de muestra para análisis final*). Los tres algoritmos fueron ejecutados en un equipo con procesador Intel Pentium 4 de 3.2GHz y 1GB de memoria RAM. A fin de ilustrar los resultados obtenidos en las corridas finales, a continuación se muestra el promedio y el mejor valor que presentó cada algoritmo en cada una de las instancias propuestas. Los valores presentados representan el Cmax (en unidades de tiempo).

Tabla 12: Resultados de los algoritmos por instancias

INSTANCIA	ALGORITMO GENÉTICO		VIRAL SYSTEM PERMUTADO		VIRAL SYSTEM RK	
	Promedio	Mejor Valor	Promedio	Mejor Valor	Promedio	Mejor Valor
1	1653,6	1560	1459,6	1299	3454,4	3296
2	730,6	670	687,8	675	2303,6	2187
3	1101	1077	1029,2	982	3032	2555
4	2289,4	2203	2141,2	2102	6317,4	5616
5	2923,8	2828	2688	2586	17772,6	17254
6	1420,2	1391	1380,6	1346	9938,4	9737
7	2020,2	2004	1985,4	1977	14124,8	13760
8	3604,8	3532	3470,2	3385	27872,2	27357
9	2528,4	2475	2386,6	2313	9328,4	9136
10	1015,4	991	966,8	930	4648,4	4009
11	1447,2	1390	1374,6	1356	6626,2	6489
12	2771,8	2711	2633,2	2545	12835,2	11762
13	2917,2	2802	2920,2	2815	9838,6	8530
14	1495,8	1464	1425,8	1406	6050,6	5826

15	2442,2	2381	2325	2197	8919,4	7538
16	5404,6	5348	5229,6	5184	18487,6	16448
17	6017,4	5824	5865,4	5773	43608,6	40638
18	2477	2431	2395	2371	26036,2	23833
19	3653,2	3612	3574,8	3525	39604,2	38932
20	7289,6	7215	7163	6827	78872,6	75122
21	5685	5571	5224,6	4545	21946,8	21559
22	1970,6	1908	1912,8	1888	13056,8	11567
23	2956,2	2916	2852,6	2767	18178,4	16082
24	6309,4	6264	6066,2	5736	39302,2	37283
25	10044,8	9881	9804	9545	30056,8	27774
26	2914,8	2905	2951,8	2924	13385	12007
27	4743,2	4710	4706,2	4588	20296,4	20073
28	10466	10342	10309,8	10089	40173,8	36916
29	11725,8	11521	11448,4	10881	112985	111293
30	4060,4	4040	4054,6	3997	55630,8	53047
31	6302	6287	6134,6	5988	85820	85287
32	12891,2	12733	12533,2	12257	170418,6	169745
33	10117,6	9977	9993,2	9769	55063,6	54353
34	3344,4	3290	3361,6	3311	27764	27343
35	5320,6	5249	5209,8	5167	40487,6	38296
36	11190,2	11012	10965,2	10353	81806,4	80045

De la tabla anterior se puede observar que el 89% de las veces el algoritmo viral system permutado encontró la mejor solución y un 11% de las veces lo arrojó algoritmo genético. De igual forma en un 92% de las instancias viral system obtuvo el mejor promedio de Makespan y el 8% corresponde algoritmo genético.

A fin de determinar y evaluar el desempeño de los algoritmos presentados en este trabajo, principalmente de *Viral System*; y verificar si existen diferencias estadísticamente significativas entre las soluciones obtenidas por uno u otro algoritmo. En primera instancia se procedió a realizar un análisis de Varianza (ANOVA) simple, sin embargo los datos generados no cumplió el supuesto de normalidad. Para el caso en el que el supuesto de normalidad no está justificado, la prueba *Mediana de Mood* es una alternativa no paramétrica que se utiliza para

probar la hipótesis nula de que la media de los tratamientos son idénticos, contra la hipótesis alternativa de que al menos un par de medianas de los tratamientos son diferentes [51].

A fin de efectuar comparaciones del desempeño de cada algoritmo independientemente de las instancias, hubo la necesidad de estandarizar el valor de la función objetivo obtenido mediante la ecuación anteriormente enunciada

$$Z_i = \frac{x_i - \bar{x}}{s}$$

Donde i representa cada una de las instancias, $\bar{x}$ representa la media y s la desviación estándar de cada una de las instancias. A continuación se muestra la prueba de *mediana de mood* realizada en el software *StatGraphics* a fin de determinar si existen o no diferencias entre los algoritmos.

Tabla 13: Prueba de comparación de algoritmos con respecto al Cmax

PRUEBA DE MEDIANAS DE MOOD PARA Z (CMAX)						
Algoritmo	Tamaño Muestra	n<=	n>	Mediana	LC inferior 95,0%	LC superior 95,0%
GENÉTICO	180	97	83	-0,676238	-0,678827	-0,673508
VSPRMU	180	173	7	-0,686133	-0,687843	-0,684847
VSRK	180	0	180	1,38087	1,36935	1,39162

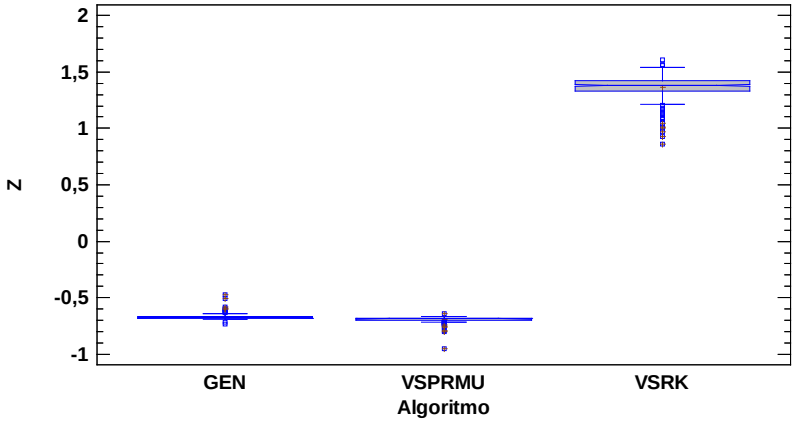
Estadístico = 334,178 Valor-P = 0

La prueba de medianas de Mood se realizó bajo un nivel de confianza de 95%, por lo que el P-Valor=0 indica que si existen diferencias significativas entre los niveles del factor, es decir los algoritmos propuestos.

La prueba de medianas de Mood indica solamente si existen diferencias entre los tratamientos de un factor, sin embargo no específica cuál de los tratamientos es

mejor según sea el caso. A fin de determinar cuál de los algoritmos evaluados provee mejores soluciones para el Problema de programación de la producción en entornos Flexible Flowshop, se escogió el análisis de los datos finales en el diagrama de Caja y Bigotes presentado a continuación, el cual señala una muesca en la mediana de los resultados obtenidos por cada algoritmo a fin de determinar cuáles medianas son significativamente diferentes de otras.

Ilustración 22: Grafico de Cajas y bigotes de comparación algoritmos con respecto al Cmax



La variable de respuesta, en este caso los datos del análisis es la respectiva normalización por instancia descrita anteriormente.

Según el gráfico de Caja y Bigotes, se nota que *Viral System* Permutado, presenta los menores valores de la variable en estudio, mientras que Viral system Random Key los valores más altos. Según la comparación de muestras de la opción de Gráficos de Caja y Bigotes de *StatGraphics,* las muescas de las cajas están en una escala tal que, para muestras de igual tamaño, si no se traslapan, las dos medianas son significativamente diferentes a un nivel de confianza del 95%; por tal razón, el gráfico permite afirmar que todos los algoritmos son significativamente diferentes entre sí, y que V*iral System* permutado provee mejores soluciones para el problema.

Además de evaluar la calidad de las soluciones en cada uno de los algoritmos, es necesario analizar el tiempo de ejecución de los mismos y las diferencias entre ellos. A continuación se muestra los resultados de la prueba *mediana de mood* para verificar si existen diferencias estadísticamente significas entre los tiempos de ejecución de los algoritmos.

Al igual que los procedimientos anteriores, la variable de salida, en este caso tiempo de ejecución, fue estandarizada por instancia. La prueba fue realizada en el software *StatGraphics* y los resultados arrojados fueron los siguientes:

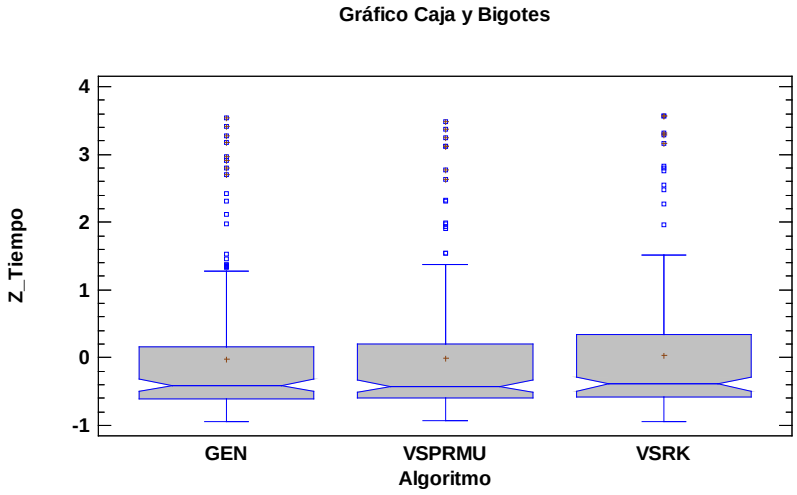
Tabla 14: Prueba de comparación de algoritmos con respecto al Tiempo

PRUEBA DE MEDIANAS DE MOOD PARA Z (TIEMPO)						
Algoritmo	Tamaño Muestra	n<=	n>	Mediana	LC inferior 95,0%	LC superior 95,0%
GEN	180	90	90	-0,410485	-0,435508	-0,334408
VSPRMU	180	94	86	-0,422109	-0,442284	-0,379857
VSRK	180	86	94	-0,390522	-0,422695	-0,316371

Estadístico = 0,711111 Valor-P = 0,700784

La prueba de *Mediana de Mood* se realizó bajo un nivel de confianza de 95%, por lo que el P-Valor=0.7 indica que no existen diferencias significativas entre los niveles del factor, es decir, en los algoritmos propuestos con respecto al tiempo de ejecución de cada uno de ellos. Para tener mayor claridad presentamos a continuación el diagrama de caja y bigotes.

Ilustración 23: Grafico de Cajas y bigotes de comparación algoritmos con respecto al Tiempo



En el gráfico de cajas y bigote se puede observar que los tres algoritmos tienen datos atípicos y variabilidad similar.

CAPÍTULO 5 CONCLUSIONES Y TRABAJOS FUTUROS

Después de haber realizado el respectivo análisis estadístico para verificar si existían o no diferencias significativas entre los algoritmos con respecto a calidad de las soluciones y tiempos de ejecución para diferentes instancias del *Problema de programación de la producción en entornos flexible flowshop,* se puede afirmar y concluir que entre los algoritmos programados; *Algoritmo Genético, Viral System con Encoding Random Key,* y Viral System con Encoding Permutado, éste último provee soluciones de mejor calidad para el problema estudiado que *Algoritmo Genético*, por consiguiente se puede decir *Viral System permutado* es un buen algoritmo para solucionar FFS superando a los algoritmos mencionados anteriormente con los cuales se contrastó. Según el gráfico de caja y bigotes para este análisis, *Viral System* Permutado presenta menor variabilidad, es decir, la mayoría de las veces ofrece la misma o una respuesta muy cercana respecto al valor de la función objetivo; sin embargo, los valores de función objetivo más altas,

es decir, las peores soluciones según el estudio, son las arrojadas por *Viral System Random Key.*

Se puede afirmar que *no hay diferencias estadísticamente significativas en el tiempo de duración de los algoritmos.*

En conclusión, después del desarrollo y análisis de los resultados de la investigación “Programación de la producción en un sistema flexible flow shop utilizando la metaheurística viral systems”, se puede afirmar que el algoritmo Viral System ofrece mejores soluciones que un algoritmo genético para el problema estudiado, a con un costo computacional similar, sin embargo es indispensable poder escoger adecuadamente el tipo de encoding (codificación) a utilizar como quedó verificado en este estudio esta es significativa en la calidad de las soluciones, para el caso de la programación de la producción en entornos flexible flowshop es recomendable utilizar un encodig permutado a uno con random key.

A fin de ampliar y complementar el desarrollo de esta investigación, se recomienda aprovechar el desempeño que evidencia del algoritmo de *Viral System* en programación de la producción en sistemas flexible flowshop para trabajar en todas las posibles variaciones del problema de scheduling ya otros problemas combinatorios.

Se recomienda hacer variaciones en los operadores de vecindad para determinar la utilización de éstos influye significativamente en la calidad de las respuestas o en tiempo computacional.

Finalmente se recomienda hacer una modificación al algoritmo Viral Systems a fin de hacerlo más completo, esto puede lograrse orientándolo hacia la solución de problemas multiobjetivo y estocásticos en el que además del makespan, se minimicen otros objetivos como sumatoria de la tardanza total, número de trabajos tardíos, costos, tiempo de flujo promedio o funciones objetivos más innovadoras.

CAPÍTULO 6 BIBLIOGRAFÍA

[1] L. Cortes, P;García, J; Muñuzuri, J;Onieva, "Viral System to Solve Optimization Problems: An Immune-Inspired Computational Intelligence Approach," *7th International Conference, ICARIS 2008*, P.J. Bentley, ed., Icaris: Springer Verlag, 2008, pp. 83-94.

[2] J.W. Herrmann, *HandBook of Production Scheduling*, Springer, 2006.

[3] M.L. Pinedo, *Planning and Scheduling in Manufacturing and Services*, New York: Springer, 2005.

[4] J. Dominguez Machuca, M. Álvarez Gil, S. Garcia Gonzalez, M. Dominguez Machuca, and A. Ruiz Jimenez, *Dirección de Operaciones*, Madrid: MCGRAW-HILL, 1995.

[5] M.L. Pinedo, *Scheduling. Theory, Algorithms And Systems*, New York: Springer, 2008.

[6] J. Jungwattanakit, M. Reodecha, and P. Chaovalitwongse, "Algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria," *International Journal of Advanced Manufacturing Technology*, vol. 37, 2008, pp. 354-370.

[7] R. Ruiz and J.A. Vázquez Rodriguez, "The hybrid flow shop scheduling problem," *European Journal Of Operational Research*, vol. Article in, 2009.

[8] J.N.D. Gupta, "Two-stage, hybrid flowshop scheduling problem," *Journal of the Operational Research Society*, vol. 39, 1988, pp. 359-364.

[9] J.A. Hoogeveen, J.K. Lenstra, and B. Veltman, "Preemptive scheduling in a two-stage multiprocessor flow shop is np-hard," *European Journal of Operational Research*, vol. 89, 1996, pp. 172-175.

[10] H. Djellab and K. Djellab, "Preemptive hybrid flowshop scheduling problem of interval orders," *European Journal of Operational Research*, vol. 137, 2002, p. 37–49.

[11] R. Korf, "Depth-first iterative-deepening An optimal admissible tree search," *Artificial Intelligence*, vol. 27, Sep. 1985, pp. 97-109.

[12] S.J. Russell, *Inteligencia Artificial. Un enfoque moderno*, Madrid: Pearson Educacion. S.A, 2004.

[13] R.E. Bellman, *Dynamic Programming*, Princeton: Princeton University Press, 2003.

[14] E.-ghazali Talbi, *Metaheuristics : from design to implementation*, Hoboken: John Wiley & Sons, 2009.

[15] M. Mezmaz, N. Melab, and E.-G. Talbi, "A Grid-enabled Branch and Bound Algorithm for Solving Challenging Combinatorial Optimization Problems," *2007 IEEE International Parallel and Distributed Processing Symposium*, Long Beach: IEEE, 2007, pp. 1-9.

[16] B. Melián, J.A.M. Pérez, J.M.M. Vega, D.D. Estadística, and I.O. Computación, "Metaheurísticas : una visión global," *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, vol. 19, 2003, pp. 7-28.

[17] F. Hiller and G. Liberman, *introducción a la investigacion de operaciones*, Mexico: MCGRAW-HILL, 2010.

[18] A. Duarte Muñoz, J. Fernández Pantrigo, and M. Carrillo, Gallego, *Metaheurísticas*, Madrid: Dykinson, 2007.

[19] L. Cortes, P;García, J; Muñuzuri, J;Onieva, "Viral systems: A new bio-inspired optimisation approach," *Computers & Operations Research*, vol. 35, Sep. 2008, pp. 2840-2860.

[20] T.B.K. Rao, "Sequencing in the order a, b, with multiplicity of machines for a single operation, OPSEARCH: Journal of the Operational Research Society of," *Journal of the Operational Research Society of India*, vol. 7, 1970, pp. 135-144.

[21] K.G. Ramaswamy, with T.s, "An extension of two machine sequencing problem," *OPSEARCH, The Journal of the Operational Research Society of In*, vol. 8, 1971, pp. 10-22.

[22] B.S. Mittal and P.C. Bagga, "Two machine sequencing problem with parallel machines," *Journal of the Operational Research Society of India*, vol. 10, 1973, pp. 10-22.

[23] G.-cheol Lee and Y.-dae Kim, "branch-and-bound algorithm for a two-stage hybrid flowshop scheduling problem minimizing total tardiness," *International Journal of Production Research*, vol. 42, 2004, pp. 4731-4743.

[24] J.N.D. Gupta, A.M.A. Hariri, and C.N. Potts, "Scheduling a two-stage hybrid flow shop with parallel machines at the first stage," *Annals of Operations Research*, vol. 69, 1997, pp. 171-191.

[25] M. Haouari, L. Hidri, and A. Gharbi, “Optimal scheduling of a two-stage hybrid flow shop,” *Mathematical Methods of Operations Research*, vol. 64, 2006, pp. 107-124.

[26] S.A. Brah and J.L. Hunsucker, “Branch and bound algorithm for the flow shop with multiple,” *processors. European Journal of Operational Research*, vol. 51, 1991, pp. 88-99.

[27] T.J. Sawik, “Mixed integer programming for scheduling flexible flow lines with limited intermediate buffers,” *Mathematical and Computer Modelling*, vol. 31, 2000, pp. 39-52.

[28] T. Sawik, “An Exact Approach for Batch Scheduling in Flexible Flow Lines with Limited Intermediate Buffers,” *Computer*, vol. 7177, 2002.

[29] R.J. Wittrock, “Scheduling algorithms for flexible flow lines,” *IBM Journal of Research and Development*, vol. 29, 1985, pp. 401-412.

[30] A.G.P. Guinet, “Textile production systems: a succession of non-identical parallel,” *processor shops, Journal of the Operational Research Society*, vol. 42, 1991, pp. 655-671.

[31] A.G.P. Guinet, M.M. Solomon, P.K. Kedia, and A. Dussauchoy, “A computational study of heuristics for two-stage flexible flowshops,” *International Journal of Production Research*, vol. 34, 1996, pp. 1399-1415.

[32] A.G.P. Guinet and M.M. Solomon, “Scheduling hybrid flowshops to minimize maximum tardiness or maximum completion time,” *International Journal of Production Research*, vol. 34, 1996, pp. 1643-1654.

[33] R. Ruiz and J.A. Vázquez-rodríguez, “The hybrid flow shop scheduling problem,” *European Journal of Operational Research*, vol. 205, 2010, pp. 1-18.

[34] S. Voss, “The two-stage hybrid flow shop with sequence-dependent setup times,” *Operations Research in Production Planning and Control, Proceedings of a Joint German/US Conference*, A. Jones, G. Fandel, and T.R. Gulledge, eds., Springer-Verlag, 1993, p. 215–220.

[35] M. Haouari and R. M’hallah, “Heuristic algorithms for the two-stage hybrid flowshop problem,” *Operations Research Letters*, vol. 21, 1997, pp. 43-53.

[36] B. Wardono, “A tabu search algorithm for the multi-stage parallel machine problem with limited buffer capacities,” *European Journal of Operational Research*, vol. 155, Jun. 2004, pp. 380-401.

[37] C.Y. Low, “Simulated annealing heuristic for flow shop scheduling problems with unrelated parallel machines,” *Computers and Operations Research*, vol. 32, 2005, pp. 2013-2025.

[38] An, “Improved simulated annealing for hybrid flowshops with sequence-dependent setup and transportation times to minimize total completion time and total tardiness,” *Expert Systems with Applications*, vol. 36, 2009, pp. 9625-9633.

[39] V.J. Leon and B. Ramamoorthy, “An adaptable problem-space-based search method for flexible flow line scheduling,” *IIE Transactions*, vol. 29, Feb. 1997, pp. 115-125.

[40] W. Xiao, P. Hao, S. Zhang, and X. Xu, *Hybrid flow shop scheduling using genetic algorithms*, IEEE, 2000.

[41] R. Ruiz and C. Maroto, “A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility,” *European Journal of Operational Research*, vol. 169, Mar. 2006, pp. 781-800.

[42] V. Yaurima, L. Burtseva, and A. Tchernykh, “Hybrid flowshop with unrelated machines, sequence-dependent setup time, availability constraints and limited buffers☆,” *Computers & Industrial Engineering*, vol. 56, May. 2009, pp. 1452-1463.

[43] An, “An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times,” *Applied Mathematics and Computation*, vol. 180, 2006, pp. 111-127.

[44] J.N.D. Gupta, R.S. Sexton, and E.A. Tunc, “Selecting Scheduling Heuristics Using Neural Networks,” *INFORMS Journal on Computing*, vol. 12, Apr. 2000, pp. 150-162.

[45] H. Wang, V. Jacob, and E. Rolland, “Design of efficient hybrid neural networks for flexible flow shop scheduling,” *Expert Systems*, vol. 20, Sep. 2003, pp. 208-231.

[46] K.-C. Ying and S.-W. Lin, “Multiprocessor task scheduling in multistage hybrid flow-shops: an ant colony system approach,” *International Journal of Production Research*, vol. 44, Aug. 2006, pp. 3161-3177.

[47] K. Alaykýran, O. Engin, and A. Döyen, “Using ant colony optimization to solve hybrid flow shop scheduling problems,” *The International Journal of Advanced Manufacturing Technology*, vol. 35, May. 2007, pp. 541-550.

[48] B.A. Norman and J.C. Bean, “A genetic algorithm methodology for complex scheduling problems,” *Naval Research Logistics*, vol. 46, Mar. 1999, pp. 199-211.

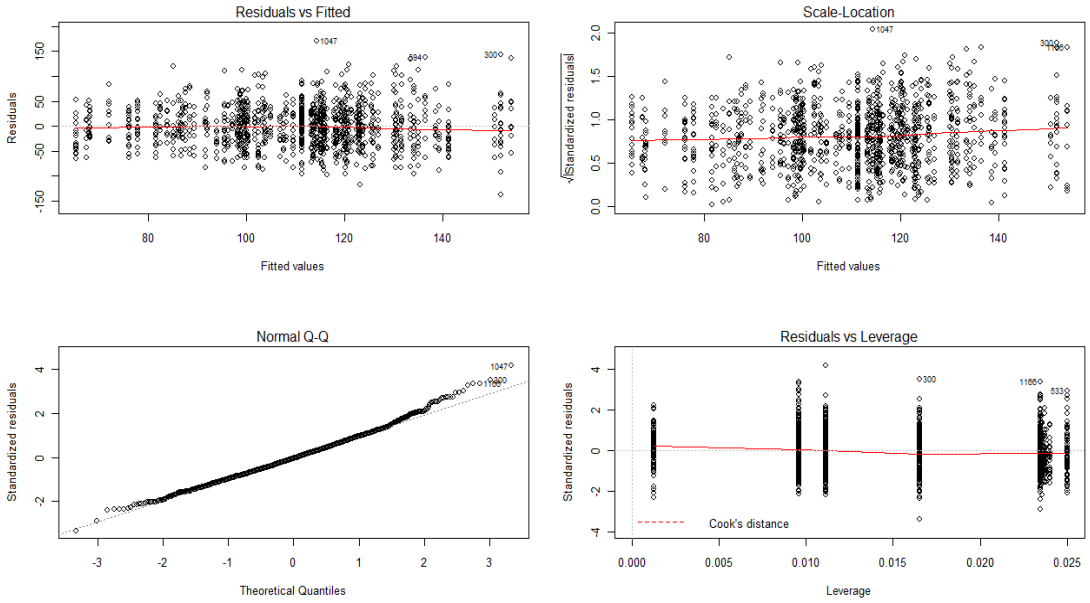
[49] M. Nawaz, E.E. Enscore Jr, and I. Ham, “A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem,” *Omega*, vol. 11, 1983, pp. 91-95.

[50] R. Ruiz, C. Maroto, and J. Alcaraz, “Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics,” *European Journal of Operational Research*, vol. 165, 2005, pp. 34-54.

[51] D.C. Montgomery, *Diseño y análisis de experimentos*, Limusa-Wiley, 2002.

ANEXOS

ANEXO 1: DIAGNOSTICO DEL MODELO DE REGRESIÓN PARA OBTENER LOS PARÁMETROS DE VIRAL SYSTEMS RANDOM KEY



Los gráficos mostrados anteriormente muestran que el modelo no tiene desviaciones muy grades con respecto a los supuestos, los cuales son Independencia de los residuales, la cual se puede apreciar en la gráfica de residuos Vs valores predichos. El supuesto de normalidad se puede observar en el grafico Normal QQ, el cual se observa que no hay una desviación significativa de este supuesto y por ultimo homocedasticidad el cual verificaremos por test de Bartlett.

Bartlett test of homogeneity of variances

data: 1 + (((EST_CMAX + 6.4)^3.1) - 1)/(3.1) by Nmax by TCC by plt2 by pr by pin2 by pan by LNR by LIT by Tol by I(Nmax^2)

Bartlett's K-squared = 5.938, df = 2, p-value = 0.05135

Como el valor de p es mayor que 0.05, Por lo tanto podemos considerar que las varianzas de los grupos son constantes.

El test de Durbin-Watson nos determina si existe auto correlación de los datos:

Durbin-Watson test

data: 1 + (((EST_CMAX + 6.4)^3.1) - 1)/(3.1) by Nmax by TCC by plt2 by pr by pin2 by pan by LNR by LIT by Tol by I(Nmax^2) DW = 2.1006, p-value = 0.9575

Como el valor P es mayor que 0.05, no se rechaza la hipótesis nula que indica que la correlación de los datos es 0.

El test de Lilliefors (Kolmogorov-Smirnov) es una prueba de bondad y ajuste que la realizamos para verificar si los datos provienen de una distribución normal:

Lilliefors (Kolmogorov-Smirnov) normality test

data: residuos D = 0.0223, p-value = 0.1693

Como el valor P es mayor que 0.05, no se puede rechazar la idea que los residuales provienen de una distribución normal

Con esto quedan verificados los supuestos para utilizar el modelo de regresión.

ANEXO 2: DIAGNOSTICO DEL MODELO DE REGRESIÓN PARA OBTENER LOS PARÁMETROS DE VIRAL SYSTEMS PERMUTADO

Lilliefors (Kolmogorov-Smirnov) normality test

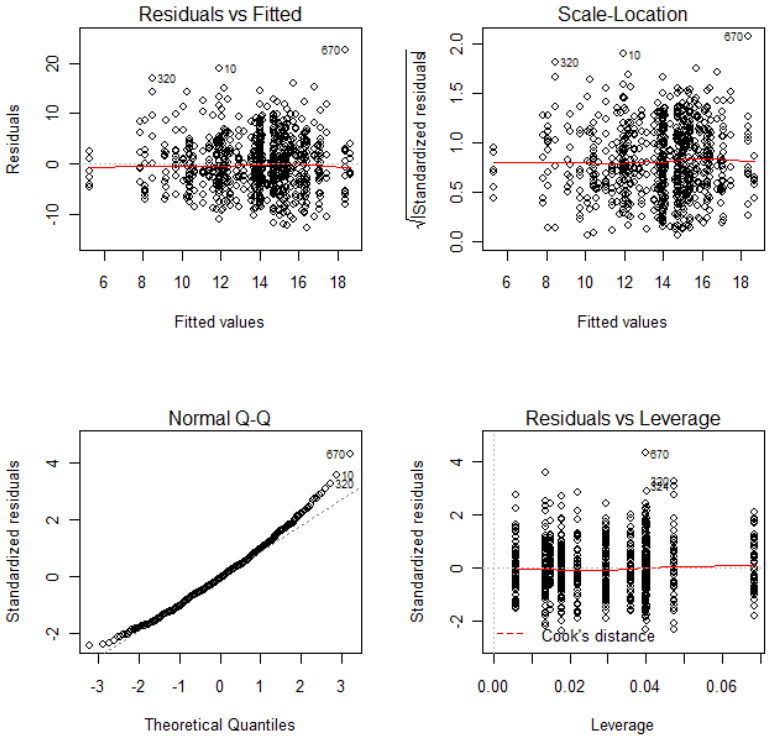
data: residuos
D = 0.0313, p-value = 0.06804

Bartlett test of homogeneity of variances

data: 1 + (((Zmax)^2.2) - 1)/(2.2) by Nmax by TCC by plt2 by pr by pin2 by pan by LNR by Tol by I(LNR^2) by I(pr^2) by I(plt2^2) by I(pan^2) by I(pin2^2)
Bartlett's K-squared = 5.9221, df = 2, p-value = 0.05177*

Durbin-Watson test

data: formula(modelo1)
DW = 1.9451, p-value = 0.2089
alternative hypothesis: true autocorrelation is greater than 0



ANEXO 3: DETERMINACIÓN DEL TAMAÑO DE LA MUESTRA.

Una decisión crítica en el diseño experimental es establecer cuál es el tamaño de la muestra, es decir cuántas replicas deben correrse.

Para determinarla se utiliza la siguiente formula:

$$\Phi^2 = \frac{n * D^2}{2 * a * \sigma^2}$$

Dónde:

Φ^2: es un indicador que se asocia con el error tipo II en la curva característica de operación.

n: tamaño de la muestra.

D: representa la diferencia entre la medias de los tratamientos en la cual la hipótesis nula deberá rechazarse.

a: número de niveles del factor para nuestro caso representa el número de algoritmos a comparar.

σ^2: Varianza de los tratamientos.

La varianza la estimamos con el cuadrado medio del error de la tabla ANOVA del algoritmo viral system permutado, dado que este representa un estadístico insesgado de la varianza poblacional.

El indicador D se calcula usando el la trasformación del modelo de regresión del algoritmo viral system permutado de la siguiente forma:

Se desea que el diseño experimental pueda rechazar la hipótesis nula si existe una diferencia entre los tratamientos de 0.5 en la variable normalizada del makespan, por lo tanto esto representa en la variable transformada= $1 + \frac{(Z+4.5)^{2.2}-1}{2.2} = 1 + \frac{(0.5+4.5)^{2.2}-1}{2.2} = 16.68$.

$$\Phi^2 = \frac{n*D^2}{2*a*\sigma^2} = \frac{n*16.68^2}{2*3*28.68} = 1.62n$$

Se utilizó una curva característica de operación con v_1=a-1=2 y v_2=a*(n-1)

n	Φ^2	Φ	a*(n-1)	β	Potencia=1- β
3	4,86	2,2045	6	0.25	0.75
4	6,48	2,5456	9	0.11	0.89
5	8,1	2,8460	12	0.03	0.97

Como requerimos que la potencia sea mayor que 0.9, por lo tanto necesitamos por lo menos 5 réplicas en el experimento.